Australian Government
Department of Defence
Science and Technology

The Vital Planning and Analysis (ViPA) ORBAT Data Service Architecture and Design Overview

Kyran Lange

Joint and Operations Analysis Division
Defence Science and Technology Group

DST-Group-TN-1539

ABSTRACT

The Vital Planning and Analysis (ViPA) workbench is an automated logistics feasibility analysis tool used to support planning and logistics. ViPA makes use of Order of Battle (ORBAT) data as an input for its calculations. The ORBAT Data Service is a Web Service for storing force structures to be used by the ADF. The service is composed into what is known as a Service Oriented Architecture which provides a loose coupling of self-contained services. This architecture helps to provide a reusable, access controlled and resilient data source. This report describes the design of the ORBAT Data Service and how it is used to manage and share ORBAT data between tools such as ViPA.

Published by

Joint and Operations Analysis Division
Defence Science and Technology Group
West Avenue
Edinburgh, South Australia 5111 Australia

Telephone: 1300 333 362
Fax: +61 8 7389 6567

AR-016-657
Updated July 2016

Approved for Public Release

The Vital Planning and Analysis (ViPA) ORBAT Data Service Architecture and Design Overview

Executive Summary

The Vital Planning and Analysis (ViPA) workbench is an automated logistics feasibility analysis tool used to support planning and logistics. ViPA makes use of Order of Battle (ORBAT) data as an input for its calculations. ORBATs describe the identification, strength, command structure, and disposition of the personnel, units, and equipment of a military force. They are complicated structures which are often captured in spreadsheets and maintained by various personnel over long periods of time in different levels of detail for a range of purposes. However, there was previously no authoritative, consolidated, shared source of ORBAT data in the Australian Defence Force (ADF).

The ORBAT Data Service was developed by the DST Group in 2008 to address requirements identified by the Logistics Planning Dedicated User Group (DUG) for modelling and sharing ORBAT data. The DUG was run and managed by Strategic Logistics Branch (SLB) for input, review and comment on the creation of user requirements for logistics operations planning tools. The ORBAT Data Service has since been deployed onto laptops that were taken into theatres overseas, installed on the Mission Secret Network (MSN) for Army exercises, and has been transitioned to the Defence Restricted Network (DRN) and the Defence Secret Network (DSN) by Defence project JP2030.

The ORBAT Data Service is a Web Service for storing force structures to be used by the ADF. Web Services are software used to share business logic and data across a network through application programming interfaces (APIs) using open standards to simplify sharing and reuse. The service is composed into what is known as a Service Oriented Architecture which provides a loose coupling of self-contained services. This architecture helps to achieve the design intent of the ORBAT Data Service to be a reusable, access controlled and resilient data source.

Part of the work in creating the service involved developing a data model for storing force structures and including the dynamic relationships between them. This included a solution for capturing the temporal dimension of force structures to represent changes in capability and force composition (e.g. unit rotations) over time. The data model caters for different types of ORBATs, namely generic capability bricks, force-in-being, hypothetical future forces, etc. Robust APIs were also designed to expose a powerful service that enables users to create, search, read, update and delete data models of ORBATS.

A comprehensive data management process was also designed and built into the service to support a rigorously managed collection of accurate, fit-for-purpose data with associated metadata.

ORBAT data services enable the Australian Defence Organisation to access authoritative ORBAT data using various applications in a number of domains including operations

planning, logistics analysis and planning, joint movement and preparedness. It allows for different instances of an ORBAT data service to be set up to manage different types of data for actual operations, specific exercises, simulations, strategic analysis, training, etc. and it is also deployable for standalone use on laptops or on fixed and adhoc networks.

This report describes the design of the ORBAT Data Service and how it is used to manage and share ORBAT data between tools such as ViPA.

Acknowledgements

I would like to thank and acknowledge Mr Shane Reschke who co-developed the ORBAT Data Service and wrote some documentation which was used to create this report.

I would also like to thank Dr Mark Nelson who read through drafts of this report and provided many helpful suggestions. His help was invaluable.

I would also like to thank both Mr Hing-Wah Kwok and Mr Kuba Kabacinski (Consunet Pty Ltd) for aiding with some of the concepts outlined in this paper. Kuba also contributed documentation which was used in the creation of this report.

Contents

This page intentionally blank

Glossary

ADF	Australian Defence Force
AM	Aide Memoire
AMDS	Aide Memoire Data Service
API	Application Programming Interface
ASJETS	Australian Joint Essential Tasks
COA	Course of Action
CONOPS	Concept of Operations
DES	Defence Entitlement System
DMO	Defence Materiel Organisation
DPR	Defence Preparedness Requirement
DRN	Defence Restricted Network
DSN	Defence Secret Network
DST Group	Defence Science and Technology Group
DSTO	Defence Science and Technology Organisation
DUG	Logistics Planning Dedicated User Group
EIS	Enterprise Information Systems
FTE	Full Time Entitlement
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
HQ	Headquarters
IBM	International Business Machines Corporation
IDA	Integrated Defence Architecture
Java EE	Java Platform, Enterprise Edition
JAX-WS	Java API for XML Web Services

JAXB	Java Architecture for XML Binding
JDBC	Java Database Connectivity
JIPB	Joint Intelligence Preparation of the Battlespace
JLO	Joint Logistics Orientation
JMAP	Joint Military Appreciation Process
JPA	Java Persistence API
JTDS	Joint Training Data Services
LE	Loan Entitlement
MLOC	Minimum Level of Capability
MVC	Model View Controller
NATO	North Atlantic Treaty Organization
ODM	Order of Battle Data Model
OLOC	Operational Level of Capability
OPO	Operational Preparedness Objective
ORBAT	Order of Battle
ORBATDS	Order of Battle Data Service
ORBO	ORBAT of ORBATs
ORM	Object/Relational Mapping
OWG	ORBAT Services Working Group
POJO	Plain Old Java Objects
RCP	(Eclipse) Rich Client Platform
RDBMS	Relational Database Management System
REST	REpresentational State Transfer
RTS	Raise, Train, Sustain
SIDC	Symbol ID Code

SLB	Strategic Logistics Branch
SME	Subject Matter Expert
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPA	Security Protected Assets
TOPFAS	Tool for Operations Planning Functional Area Service
UE	Unit Entitlement
UI	User Interface
UML	Unified Modeling Language
ViPA	Vital Planning and Analysis
WUC	Weapon User Category
XML	Extensible Markup Language
XPA	eXtreme Programming Activity
XSL	Extensible Stylesheet Language
XSLT	XSL Transformation

This page intentionally blank

1. Introduction

The Vital Planning and Analysis (ViPA) workbench is an automated logistics feasibility analysis tool used to support planning and logistics. ViPA makes use of Order of Battle (ORBAT) data as an input for its calculations. ORBATs describe "The identification, strength, command structure, and disposition of the personnel, units, and equipment of any military force."[22] They are complicated structures which are usually captured in spreadsheets and maintained by various personnel over long periods of time in different levels of detail for a range of purposes. However, there was previously no authoritative, consolidated, shared source of ORBAT data in the Australian Defence Force (ADF).

The ORBAT Data Service (ORBATDS) was developed by the Defence Science and Technology Group (DST Group) in 2008 to address requirements identified by the Logistics Planning Dedicated User Group (DUG) for modelling and sharing ORBAT data. The DUG was run and managed by Strategic Logistics Branch (SLB) for input, review and comment on the creation of user requirements for logistics operations planning tools.

The ORBATDS is a Web Service which acts as a storage facility for different types of ORBATs including generic force structures (i.e. capability bricks), actual force structures (i.e. force-in-being and unit entitlements) and capability lists for Defence Preparedness Requirements (DPRs).

Web Services are software used to share business logic and data across a network through application programming interfaces (APIs) using open standards to simplify sharing and reuse. The service is composed into what is known as a Service Oriented Architecture (SOA) which provides a loose coupling of self-contained services. This architecture helps to achieve the design intent of the ORBAT Data Service to be a reusable, access controlled and resilient data source. The purpose of the service is to store these various ORBATs and control access across different applications. Thus applications can consume these ORBATs and modify them locally within the application to satisfy their business requirements or alternatively applications can consume, modify and produce ORBATs for storage back into the service.

A particularly challenging aspect not handled before is how force structures change over time. Force structures in defence change for a number of reasons including through unit rotations while on deployment, with the introduction of new capabilities (e.g. new equipment) and with the Defence White Paper which provides guidance about Australia's long-term defence capability. By adding a temporal dimension to the data stored in the ORBATDS it is possible to model the evolution of the force over time and then to use planning and simulation tools to analyse the impacts of that change on ADF capability in various scenarios.

Historically the ADF has used a variety of methods for maintaining ORBATs. Not all of these are formalised and included the use of notebooks, spreadsheets and presentation slides, with email and formatted military messaging used to distribute ORBAT information. There are some problems with this haphazard method of maintaining

ORBAT data. Firstly, by not having a centralised source of truth it is difficult to know whether the information you currently have is up-to-date. Because copies are getting emailed and forwarded, possibly with modifications, there is no easy method to know how correct the version you are looking at is. This leads to wasted effort in determining the correctness of the information and makes collaboration very difficult.

The ORBATDS addresses many of these problems. As it is a web service it is not tied to any particular user ORBAT editing application, thus allowing potentially multiple user interfaces to be created for specific purposes. The SOA based design more easily accommodates the uptake of new information technologies. The ORBATDS has been designed to allow for multiple services to co-exist making it scalable for different situations. By following open standards it is easily interoperable with other applications which assists in information sharing and openness. Being a Web Service also assists with collaboration. Multiple people can interact with the service simultaneously using it as a platform for collaboration. In addition, it meets the need for a centralised authoritative data source of force information. When users access data from the service they can be guaranteed that it is the most accurate and up-to-date representation of the force that is available.

ORBAT data services enable the Australian Defence Organisation to access authoritative ORBAT data using various applications in a number of domains including operations planning, logistics analysis and planning, joint movement and preparedness. It allows for different instances of an ORBAT data service to be set up to manage different types of data for actual operations, specific exercises, simulations, strategic analysis, training, etc. and it is also deployable for standalone use on laptops or fixed and adhoc networks.

The ORBATDS has been transitioned to the Defence Restricted Network (DRN) and the Defence Secret Network (DSN) by Defence project JP2030. The ORBATDS has also been deployed onto laptops that were taken into theatres overseas, installed on the Mission Secret Network (MSN) for Army exercises, and is attracting interest from other parties.

1.1 Related Work

The Australian Army have a system called the Defence Entitlements System (DES) which is used to manage an instance of ORBAT data called Unit Entitlements (UEs). This has a command-line interface and has maintenance issues due to the small number of people who are familiar with this legacy system. It acts as a centralised repository for the UEs, but handles information sharing poorly as the UEs are exported into a spreadsheet which is then distributed. This causes the same problems as mentioned above. The data is also difficult to then use in other applications due to the lack of open standards.

There are a number of foreign tools used to assist with the management of, and facilitate the use of, ORBAT data. The Tool for Operations Planning Functional Area Services (TOPFAS) provides data and planning support tools for North Atlantic Treaty Organization (NATO) operational planning. It is developed by the NATO Consultation,

Command and Control (C3) Agency, together with partners from industry since before 2001 [29]. TOPFAS has a component called the ORBAT Management Tool (OMT) which is used to populate the TOPFAS database with the ORBAT to support the operational planning process. The OMT, similarly to the ORBATDS discussed in this report, can support the representation of real units as well as generic units and allows for the breaking down of units into command structures. TOPFAS outputs are Microsoft PowerPoint and Microsoft Word documents. [26, 27]

Another related work is the Joint Training Data Services (JTDS) which is sponsored by the United States of America Department of Defense Joint Staff J7. It is a web-based set of services that provide modelling and simulation ready data and scenario development tools to support theatre level constructive training [23]. JTDS comprises three services [24], one of which is the Order of Battle Service (OBS). The OBS provides distributed editing and validation with user assignable permissions, a web based data repository and the ability to 'drag and drop' entities to form a scenario.

Prior to 2005 an ORBAT Services Working Group (OWG) was established involving Defence Materiel Organisation (DMO), DST Group and Defence Industry staff. The OWG role was to ensure that ORBAT tools and ORBAT data used in multiple Australian Defence systems interoperate to a level acceptable to the Command Support Systems Branch of the DMO. The OWG met formally on four occasions and the results were documented by Coomber et. al. [11]. During the meetings, concepts, use cases and specifications were evolved through the development of a number of prototypes by both DST Group and industry. This experience was distilled as a detailed design for ORBAT services including definitions of it's data model, interface and technologies. The ORBATDS shares some of the design developed by the OWG, most importantly by being a Web Service. In addition, the ORBATDS data model shares many similarities and the technologies used such as XML, WSDL and SOAP make it compatible with their design. XSLT could be used to share data between the data models making the ORBATDS interoperable with the specification developed by the OWG.

One of the ways the ORBATDS differs from the OWG specification is it's adoption of the MIL-STD-2525B w/CHANGE 2 symbology standard [31]. Military symbology is used widely within the military to denote ORBATs and their capability so it is important for this information to be captured and used by the ORBATDS. Unfortunately various aspects of symbology have been adopted inconsistently across the ADF making finding a suitable standard to implement difficult. Based on army doctrine publications, LWP-G 0-1-5 Military Symbology is the doctrinal publication for the military symbols, which unfortunately at the time was not yet published with the previous publication being from 1997 and marked as obsolescent. Minutes of the 2-09 Joint Operations Doctrine Management Group show that APP-6 [44] and MIL-STD-2525 [31] are supported by the Attorney General and by Defence. ADFP 103 [45] doctrine also covers military symbology, with Part 2 defining ADF specific unit size symbology which was one of the points of contention for adopting MIL-STD-2525B, but the minutes noted previously indicate that this specification will not be used post 2010. A Defence Symbology Stakeholders Group was established in 2008 in order to update ADFP 103 and establish and maintain a readily

accessible glossary of symbols after determining the extend of symbology used in Defence and organisations linked to Defence. No meeting minutes from this group could be found. The Army doctrine LWD 5-1-1, Staff Officers' Guide, 2007 has a section on operational symbology which is based on MIL-STD-2525B. Based on the information found and desired interoperability with coalition partners, MIL-STD-2525B was chosen for use by the ORBATDS.

In 2003 a VCDF directive [30] initiated an evaluation of logistic support arrangements for operations conducted in the Middle East Area of Operations. This culminated in what is known as the Hingston Report [25] which provided an evaluation of ADF logistics support to operations in the middle east with a view to informing future logistic capability development. This report identified that there was a lack of competent logistics analysis tools to inform planning for operations and that their development should be advanced as a matter of priority. This prompted the development of ViPA and the ORBATDS.

The CIOG Integrated Defence Architecture (IDA) Technology Stack, pictured in Figure 1, was developed as a means to guide and align future Defence investment decisions in information technology [28]. One of the core tenets of this architecture is the Service Oriented Architecture Backbone with services providing functionality useful to Defence in an interoperable, modular and reusable package. The ORBATDS itself is a Web Service which fits into the Data and Infrastructure layers of the IDA. An administration client was built to manage the data stored in the ORBATDS and fits into the Data layer of the technology stack. The ViPA application, which utilises the Web Services, aligns with the top three layers. By fitting within the CIOG IDA Technology Stack this should future proof these tools.

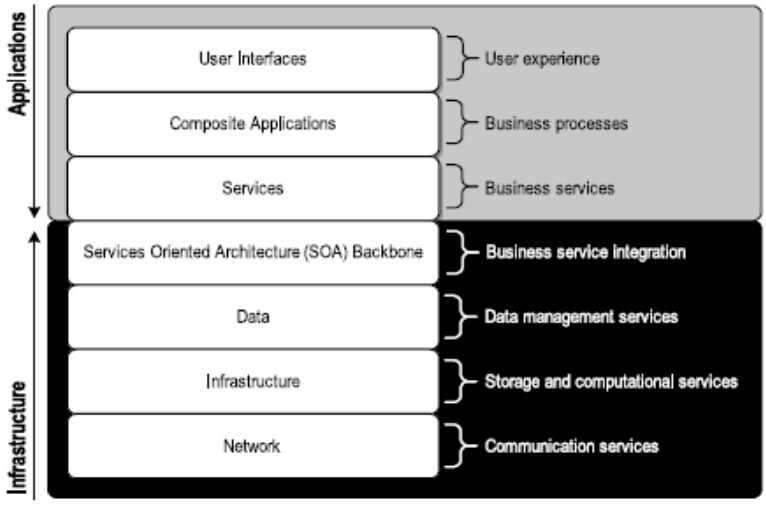

Figure 1: The layers of the CIOG Integrated Defence Architecture (IDA) technology stack

1.2 Purpose

This report gives a detailed overview of the design of the ORBAT Data Service (ORBATDS) and how it is used to manage and share ORBAT data for use between tools such as ViPA. It consolidates the requirements addressed by the service as well as documenting design choices and the technologies that were chosen. It also describes the data model and behaviours of the service to give readers a comprehensive account of the ORBATDS and give readers a thorough understanding of the service.

1.3 Intended Audience

This document is intended for those who are interested in representing and sharing ORBATs and for developers who want to integrate with the ORBATDS or wish to develop other data services using the same standards, designs and technologies.

In addition, it is hoped this document will improve awareness of the ORBATDS and its capabilities to increase its adoption within Defence and avoid duplication of effort to build a similar service.

1.4 Document Structure

Section 2 of this document gives a brief history and overview of the Vital Planning and Analysis (ViPA) project which was built in conjunction with the ORBATDS. ViPA was the initial consumer of the ORBATDS and stimulated many of the requirements for the service. Section 3 outlines the requirements and use cases for the service. The structure of the remainder of this document addresses each layer shown below in Figure 2. This structure is similar to that of the IDA stack, starting with the service interface and gradually delving deeper into the supporting detail.

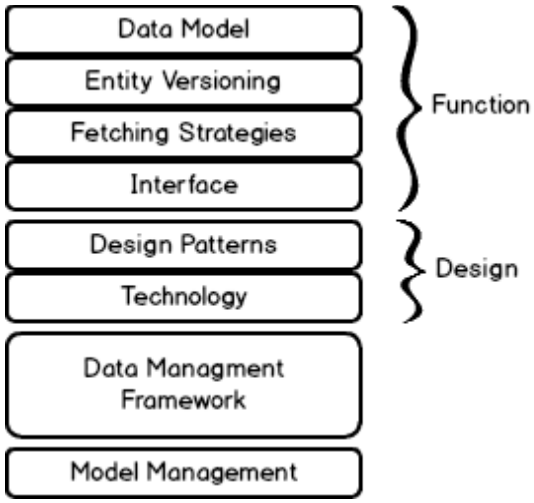

Figure 2: An overview of how the rest of this document is structured

Sections 4 through 7 describe the high-level functions of the service. Firstly the data model is dissected to give the reader knowledge on what data is recorded and how it is structured and manipulated. Following this the entity versioning is explained along with the different fetching strategies that are available. Then the application programming interfaces (APIs) that support searching and accessing the data efficiently and the administration interfaces used to manage the data within the service are described.

Section 8 focuses on the design of the service, outlining the common design patterns that have been adopted and the technology choices that were made.

Section 9 outlines the technical data management framework that is implemented by the service and explains how it behaves in certain use cases. This also includes information on how the service supports the deployment of multiple instances on the same network and/or different networks for various purposes.

Non-functional aspects of the service are outlined in section 10. This includes information about the security model employed by the service, performance limitations, the concurrency model and how the data is validated, audited, persisted and mapped.

Finally, opportunities for future development of the service are outlined in section 11.

2. ViPA

2.1 Background

The Planning and Logistics Group at the DST Group began developing ViPA in 2006 through a series of prototypes supported by user centred design activities with ADF sponsors and the end-user community. The purpose of these prototypes was originally to assist in the development of user requirements. However, due to shortfalls in current operational capability and no suitable commercial products being available on the market, these prototypes were expedited for limited interim use, as ViPA version 1.1. ViPA 1.1 shipped with the Aide Memoire Data Service (AMDS) which stores, maintains and allows network access to logistic planning data. The AMDS data included equipment, people, containers, supply items, organisations, facilities and their associated dimensions, components, consumption models and production models.

In 2008 development of ViPA 2.0 began by building upon ViPA 1.1 functionality, refining and improving the existing tools, and incorporating additional functional components which added value to the ADF logistics planning process [20]. This included moving the organisation component of the AMDS into its own data service to become the ORBAT Data Service (ORBATDS).

2.2 Support to Planning

The ViPA workbench provides automated decision support to operational and strategic level logistics planning in the areas of sustainment and distribution. The Joint Military Appreciation Process (JMAP) is the key decision-making model used in the ADF by operational and strategic level planners and is described in ADFP 5.0.1 [18]. It consists of four steps, shown in Figure 3:

1. Mission Analysis – determine which tasks are necessary to fulfil the mission. This step ensures that there is a clear understanding of the commander's intent and assists with identifying the mission and associated tasks that are essential to successfully satisfying that intent and achieving the end state.

2. Course of Action (COA) Development—refine the commander's guidance and themes into developed COAs. The COA should provide the commander with a range of workable options that can be analysed and further developed.

3. COA Analysis—test the advantages and disadvantages of each COA. This is done by analysing the friendly COA developed in step 2 against adversary COAs using a selected war gaming method.

4. Decision and Concept of Operations (CONOPS) Development—decide upon the optimal COA and develop a CONOPS for commander approval. The strengths and weaknesses of each viable COA identified during COA Analysis are compared to determine which has the highest probability of success.

As resource limitations have the potential to limit the options available to commanders, logistic planners must determine what is possible within the operational constraints while supporting the commander's intent. ViPA 2.0 allows ADF logistics planners to perform broad logistic COA analysis including comparing relative efficiencies and effectiveness of alternative distribution arrangements. With the data contained within the AMDS, ViPA also supports the Mission Analysis phase of the JMAP by providing ready access to planning data required in the development of the Joint Logistics Orientation (JLO) [20], which is described in Annex A of ADFP 4.2.2 Chapter 3 [19]. The JLO is the logistic intelligence process that supports the Joint Intelligence Preparation of the Battlespace (JIPB) which itself is a systematic, dynamic process for analysing the environment and advesarary and is also a processing medium through which intelligence staff can provide an assessment of environmental effects on an operation.

The high level context for ViPA is shown in Figure 3. Mission parameters for the current COA under consideration together with various environmental factors are provided as inputs to ViPA. The data services provide quality, authoritative data to the workbench which calculates the sustainment for that COA as well as feasibility of the movement and distribution plans to support it.

Military decision making is both an art and a science. Necessarily, the planning process is a problem solving task where solutions are being designed concurrently to new

information being sourced and challenges discovered. Thus the application of the JMAP for non-trivial problems will tend to be non-sequential and jump around across the different steps. This makes it difficult for logisticians as they endeavour to keep up with what new schemes of manoeuvre are being contemplated by planners. ViPA better enables logisticians to keep up with fast changing plans while being able to quickly analyse what the planners are creating and to do so on the basis of rigorous computational analysis of authoritative ORBAT and Aide Memoire data.

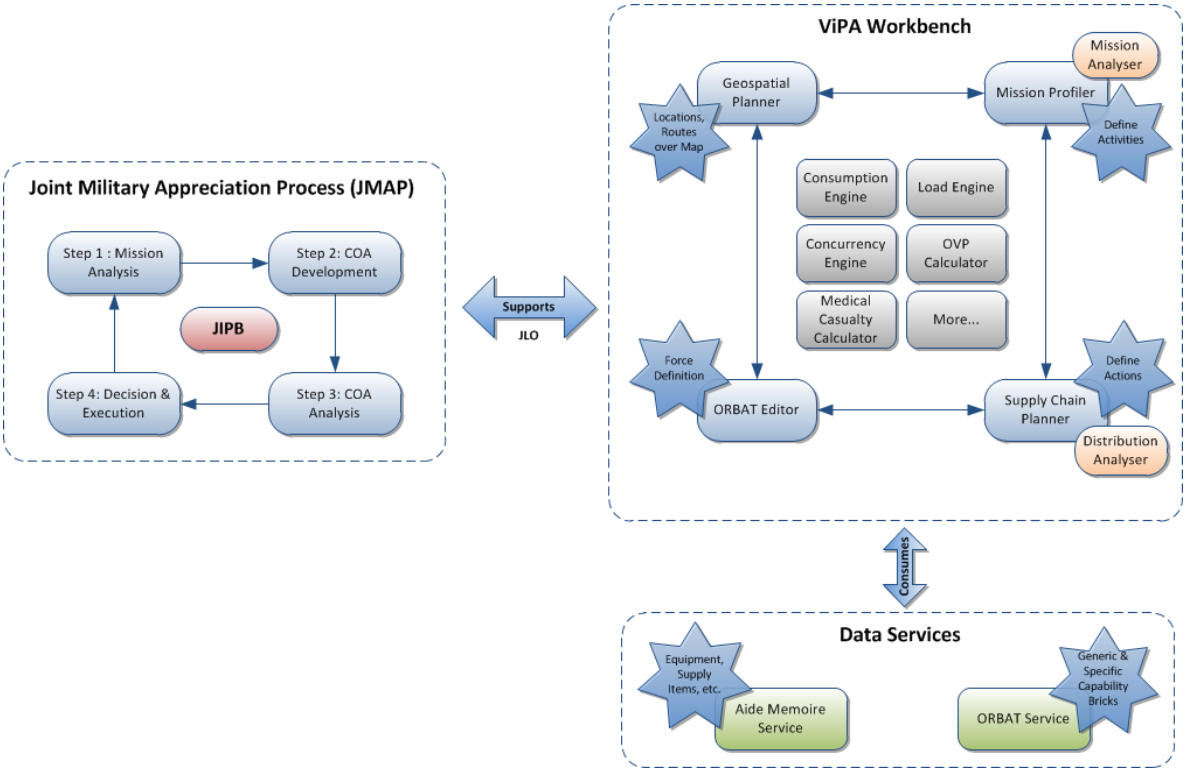

Figure 3: ViPA is used to support the Joint Logistic Orientation (JLO) which is the logistic intelligence process that supports the Joint Intelligence Preparation of the Battlespace (JIPB) that prepares the foundation for the JMAP. The four step JMAP process is shown on the left and the primary tools within the ViPA workbench which support the JMAP are shown on the right. The workbench consumes quality, authoritative data from the data services which is used to model and analyse different COA.

2.3 Architecture

ViPA has a multi-tier architecture as shown in Figure 4. This architecture is known as a 'client-server architecture' where tasks are partitioned between the providers of a resource, the data services, and the service requesters, the ViPA workbench. The first tier of the VIPA architecture is the Client Tier which contains the Presentation layer and the Application layer.

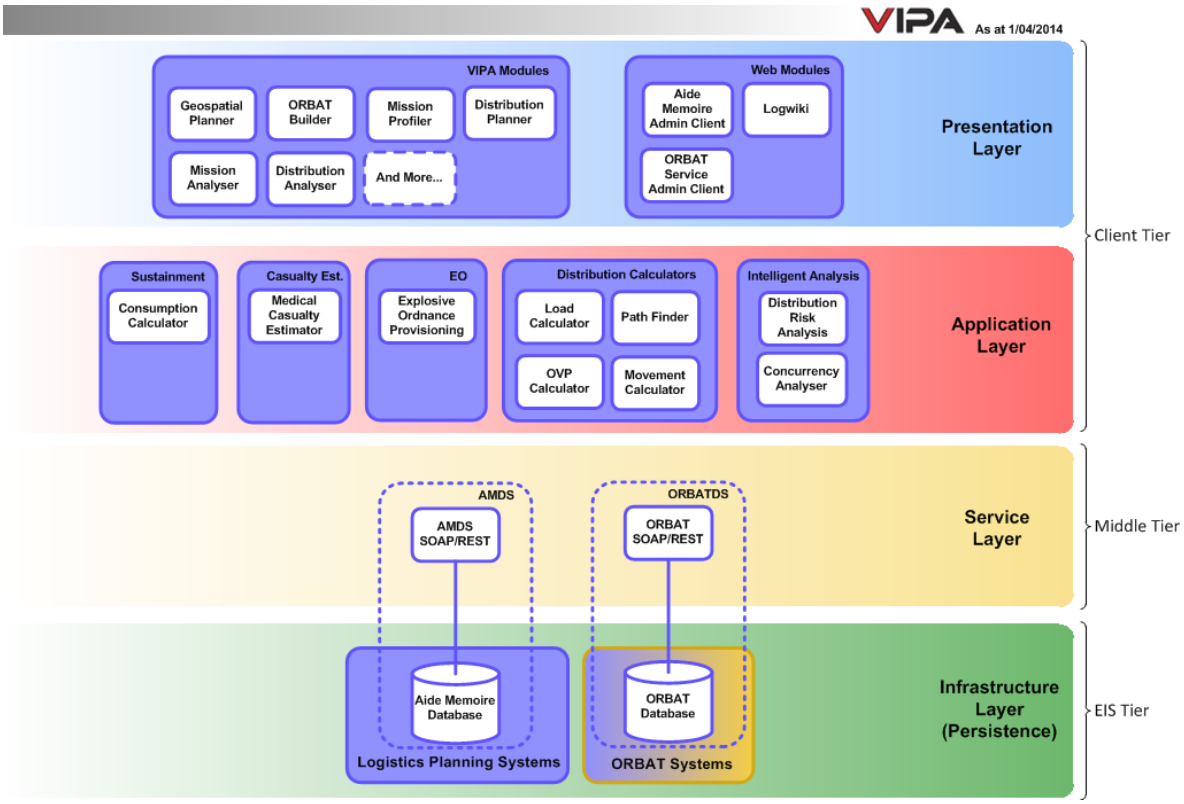

Figure 4: Overview of the ViPA multi-tier service oriented architecture

The ***Presentation Layer*** includes the interfaces that users use to interact with the system. The system includes both thick and thin client user interfaces. The ViPA workbench utilises IBM's Eclipse Rich Client Platform (RCP) which is a mature programmer tool which makes it easier to integrate independent software components. It provides a framework for the application including many user interface components. The AMDS and ORBATDS have their own thin clients for data service administration which are web applications that utilise Java Enterprise Edition (Java EE) and a number of JavaScript libraries.

The ***Application Layer*** contains engines and utilities which are invoked by the presentation layer. It also contains some functional business logic processing which provides coordination between the presentation layer and the service layer. The modules are used to process commands, make decisions, perform calculations and make evaluations. Each of the modules is written as a 'plug-in' which is a self-contained functional piece of software loaded into the RCP. The software components communicate with each other via a core-controller, thus the plug-ins don't interact directly with the presentation layer but via the controller, thus implementing the model-view-controller (MVC) architectural pattern within the client tier.

The **Middle Tier** of ViPA contains the ***Service Layer*** which provides the AMDS and ORBATDS Web Services that use open standards to allow for interoperability and to

promote re-usability. The administration clients for the AMDS and ORBATDS in the presentation layer are used to maintain the data in those services and ensure it is fit for use in the ViPA Workbench.

Finally the **Enterprise Information Systems (EIS) Tier** contains the ***Data Layer*** which is used by the Web Services to store and persist the data in databases.

3. Main Requirements

The Logistics Planning Dedicated User Group (DUG) was consulted on user requirements for the ORBATDS in eXtreme Programming Activities (XPAs) which are requirements workshops held with stakeholders. XPAs are where prototypes were demonstrated and feedback gathered to provide a large set of requirements. Their goal is to enable software development to be more user-centred and agile than traditional engineering approaches such as the waterfall model. By demonstrating prototypes it allows the users to refine their ideas about what they want, making it a useful exercise. Some of the problems identified in existing ORBAT handling solutions led to some further requirements for the ORBATDS such as the need for there to be an authoritative source of fit for purpose ORBAT data. After consolidating and rationalising the main functional requirements that were implemented in the ORBATDS we were left with the list below, as captured in the ViPA 2.0 System Specification [20].

1. Provide ORBAT and unit templates: The ORBATDS is to provide template ORBATs and Units which can be used to build up a force structure.

2. Support different types of Force Structures: Different areas of Defence use force structures in different ways so the ORBATDS should allow different types of ORBAT to be represented in the service, perhaps with different structural rules. For example, allowing the storage of capability bricks and unit entitlements. In addition, the service should allow the modelling of different national forces (e.g. Red Force).

3. Allow force structures to be modelled at different levels of abstraction: The service should support different fidelities of modelling depending on how much detail the user wants to store.

4. Store/retrieve properties about ORBATs and units: The ORBATDS needs to store and retrieve properties about ORBATs and units including name, formal name, description, time period, type, primary capability, battle dimension, echelon, affiliation, symbol (2525B) and jurisdiction.

5. Store/retrieve relationships between entities: The ORBATDS needs to store relationships between units within an ORBAT. This should allow multiple types of relationships such as command, support and maintenance. The service should also

allow storing relationships between different ORBATs to form up larger structures and reduce the overall maintenance overhead.

6. Store/retrieve Aide Memoire components which belong to a unit: Units within the ORBATDS should be allocated persons, equipment and containers from the Aide Memoire Data Service (AMDS). The ORBATDS should remain loosely coupled with the AMDS such that a connection to the AMDS is not required for the ORBATDS to be usable. The AMDS data is used to determine what an ORBAT 'has', that is, its units' load outs.

7. Capture the temporal dimension of force structures: Units are rotated and capability changes over time which needs to be captured in the ORBATDS.

8. Incorporate doctrinal symbology: The service should incorporate the military standard 2525B (with Change 2) into the data to allow for accurate unit representations. The capabilities should be informed by the standard [31].

9. Ability to search for ORBATs and units: The ORBATDS needs to provide the ability for users to search the service in order to find the ORBAT or unit that they need. The search should allow flexible searching such as searching across multiple fields or temporally to find entities within a particular time frame.

10. Incorporate a data management framework: To support the collection of data the service should have a data management framework which allows for the verification, approval and publishing of data. This is intended to improve the data quality.

11. Provide data validation: The service should have an autonomous data validation framework in order to ensure the data is fit for purpose.

12. Have a reporting framework: The service should have a framework for generating reports on the data for data management purposes. The reports should include results from the data validation (requirement 10).

13. Determine access privileges based on user roles: Users of the ORBATDS are assigned roles which are used to determine what administrative actions they are allowed to make. The ORBATDS enforces this security policy only allowing actions based on the roles: user, editor, verifier and approver. In addition, users and data should be able to be segmented into jurisdictions such that only users from a certain jurisdiction such as army, navy, joint, etc. can edit data with a matching jurisdiction.

14. Entities should be versioned: As changes are made to an entity a new version should be created so that the changes are not lost. The service should maintain a complete map of the data updates and provide temporal connectivity.

15. No deletion allowed: The ORBATDS should not allow users to delete entities, instead they should be deprecated so the historical data is not lost. This also makes

sure that people who reference a particular entity at a particular time in the service will always be able to find it. The exception to this is unpublished drafts which can be deleted as these are unable to be referenced and so can safely be deleted.

16. Provide a SOAP interface: The ORBATDS should provide a SOAP 1.2 [32] compliant interface, published via WSDL 1.1 [33], to interact with the ORBATDS. This includes things such as get, search, create, edit and deprecate functions.

17. Allow multiple instances of the ORBATDS: The ORBATDS should provide a mechanism to support multiple different ORBATDS instances being used simultaneously on a network. It will do this by preventing the local modification of entities that were imported from a different ORBAT data repository in which they were initially created.

As well as these somewhat ORBATDS specific requirements there are also system requirements to have a flexible, interoperable and extendible architecture. The data service needs to be designed for collaboration, for example military planning requires collaboration within a multi-disciplinary environment, which is important for a good operational outcome. It is also important for other applications to be able to integrate and reuse the functionality provided by the service, which means developing APIs based on open standards and technologies. Finally, it should be extensible so that custom extensions or adaptors can be built around the service architecture. These requirements are recorded in the ViPA 2.0 System Specification in section 3 [20].

Table 1 lists a number of high-level use cases which were developed for the ORBATDS to indicate how it can be used.

Table 1: High level use cases for the ORBATDS

Use Case	Title	Scenario
1	Create a new ORBAT	The user creates a new ORBAT by defining the building blocks (units) and their load out (Aide Memoire components) and then creating a command hierarchy between them to form an ORBAT. The user is able to select whether the ORBAT they are creating represents a force-in-being such as a UE or a capability brick. Any data being stored should automatically be validated to ensure it is usable.
2	Find an ORBAT	The user performs a search using filters such as type, name and capability to find a suitable ORBAT. A list of matching entities is returned to the user.
3	Retrieve an ORBAT	The user retrieves an ORBAT from the service. The retrieved ORBAT is returned in an open structured format so it can be interpreted for use in other applications.

Use Case	Title	Scenario
4	Modify an ORBAT	The user updates an ORBAT's structure and details to keep it relevant. The user can modify its structure such as adding or removing units or updating the command hierarchy. Unit's can also be modified by changing their properties or the Aide Memoire load-out. Changes made to a unit are automatically propagated up to any ORBATs which contain that unit. Major changes to an ORBAT can be captured as a new temporal version, such as when there is a major change in capability. Versions and revisions of an ORBAT can be navigated.
5	Multiple ORBATDS	The user sets up an ORBATDS for an exercise. They populate the ORBATDS with ORBATs relevant for the exercise. The data created in the service is later merged into another service without overwriting or corrupting any data in the other service.

4. Data Model

A data model is a specification for how data is defined in a system and how the data elements relate to one another. Thus a data model explicitly determines the structure of data and serves a number of purposes. Firstly, it is used to describe how the real word entities which the system is representing, in this case ORBATs, are defined in the service. In this way it can be thought of as a translation between reality and the system. Secondly, it allows users to understand the 'language' used by the service which enables them to understand and interact with the service.

The structure and fields in the data model were evolved over time to meet the requirements and use cases defined in Section 3.

An ORBAT describes "The identification, strength, command structure, and disposition of the personnel, units, and equipment of any military force."[22] There are many relationships between the components of an ORBAT and so the structure is best referred to as a graph (or network) rather than as a tree (or hierarchy). In fact, the graph for an ORBAT can contain different relationships that link multiple trees, for example, for 'Command', 'Support' or 'Maintenance' relationships between military units in the ORBAT. ORBATs are used to represent an armed force participating in activities such as raise, train, sustain (RTS), current operations or a potential future contingency. Knowing how a force is made up is important for a variety of tasks such as planning, movement, preparedness, etc. For example, the force structure can be used to determine the sustainment which will be required for a deployed force.

Figure 5 shows the ORBAT Data Model (ODM) as a UML 2 [46] class diagram. The ODM is used to represent the graph structure of ORBATs needed to address requirements in Section 3 and to support the functionality described in the following sections. The data model is also defined in an XML schema which can be found in Appendix A. The terminology used in the data model is explained below.

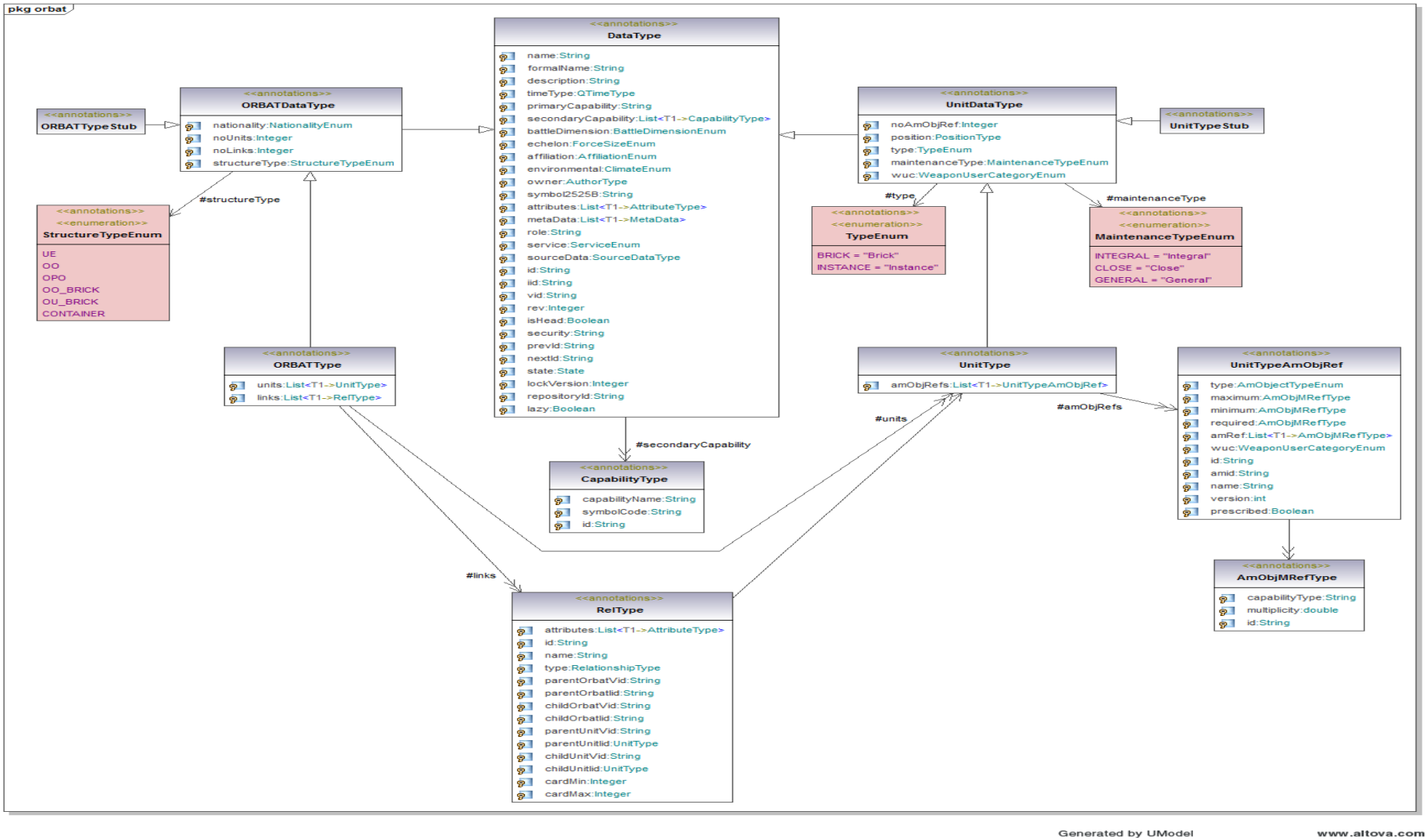

Figure 5: A UML 2 class diagram of the ORBAT Data Model used to represent ORBAT data in the ORBATDS

The two main classes are `ORBATType` and `UnitType`, where an ORBAT (i.e. an instance of `ORBATType`) contains one or more units (i.e. instances of `UnitType`). All other classes flesh out the detail of these two classes and the relationships between them. `DataType` is the common type which captures fields that are common between `ORBATType` and `UnitType`. The common fields are:

- `id`, `iid`, `vid`, `rev`, `prevId`, `nextId` are identifiers used to identify individual entities (i.e. units and ORBATs) and are also used in versioning and revisioning of the entities. More information about how these are used can be found in Section 5.
- `isHead` is used in versioning to indicate which revision is most up-to-date and hence the 'head' of the revision tree.
- `name` is the short informal name of an entity and often includes abbreviations and acronyms for example '1 BDE' or '5 RAR'. This short name is used when there is little space to display such as in reports and diagrams.
- `formalName` is the formal name of the entity that tends to be longer as it expands any abbreviations and acronyms. For example '1 Brigade' or '5 Royal Australian Regiment'.
- `description` is used to provide a description of the entity and to also capture any useful information that isn't covered by other fields.
- `timeType` is used to capture the time dimension of an entity. This describes the period of time when a particular version of an entity is valid and so enables the modelling of historic and future ORBATs. More information on how this is used can be found in Section 5.
- `primaryCapability` defines the main operational role or task for which the entity is intended to perform. Where possible this should be taken from MIL-STD-2525B with Change 2.
- `secondaryCapability` allows for a list of secondary roles or tasks to be defined. Each capability can be associated with a MIL-STD-2525B symbol.
- `battleDimension` defines the primary mission area for the war-fighting entity within the battle-space. If the battle dimension can not be or has not been determined it is considered to be unknown. If the battle dimension is known, an entity can have a mission area above the earth's surface (i.e., in the air or outer space), on the earth's surface, or below the earth's surface. If the mission area of an entity is on the earth's surface, it can be either on land or sea. The ground dimension includes those mission areas on the land surface and is divided into units, equipment, and installations. The sea surface dimension includes those entities whose mission area is on the sea surface, whereas the subsurface dimension includes entities whose mission area is below the sea surface. More information can be found in the MIL-STD-2525B standard [31].
- `echelon` indicates the command level or size of the entity e.g. squad, section and platoon.
- `affiliation` refers to the threat posed by the entity. The basic affiliation categories are unknown, friend, neutral and hostile.
- `symbol2525B` is used to store a code that identifies the military symbol or icon for an entity to be used in visualisations such as on maps or in ORBAT wire diagrams (i.e. organisational charts). A symbol ID code (SIDC) is a 15-character

alphanumeric identifier that provides the information necessary to display or transmit a tactical symbol between MIL-STD-2525 compliant systems.
    - Position 1, coding scheme, indicates which overall symbology set a symbol belongs to.
    - Position 2, affiliation, indicates the symbol's affiliation.
    - Position 3, battle dimension, indicates the symbol's battle dimension.
    - Position 4, status, indicates the symbol's planned or present status.
    - Positions 5 through 10, function ID, identifies a symbol's function. Each position indicates an increasing level of detail and specialisation.
    - Positions 11 and 12, symbol modifier indicator, identify indicators present on the symbol such as echelon, feint/dummy, installation, task force, headquarters staff, and equipment mobility.
    - Positions 13 and 14, country code, identifies the country with which a symbol is associated. Country code identifiers are listed in the Federal Information Processing Standard (FIPS) Pub 10 [47] series.
    - Position 15, order of battle, provides additional information about the role of a symbol in the battlespace. More information can be found in the MIL-STD-2525B standard [31].
- `environmental` is used to describe the climatic conditions where the entity is physically located. It uses an Enumeration of climate types from Keoppen's climate classification [43].
- `owner` is used to record who is responsible for a particular data entity. It is used in the Data Management **Framework** (see Section 9). This includes their name, user name, email address and jurisdiction.
- `state` is used the indicate the state of the entity data within the Data Management Framework (see Section 9). For example if the data is in a draft state or a published state.
- `attributes` is a list of attribute triples (name, value, vocabulary) for the entity.
- `metaData` is a list of metadata which is used to capture the author, last modified time, source and version information. It also captures the authority under whom the change has been made. It includes a comment which best practices dictate should be completed by the change author for every change.
- `role` is typically tied to specific force elements based on 'proficiencies' from Australian Joint Essential Tasks List (ASJETS) and is used to find entities which meet the requirements to fulfil a specific role.
- `service` defines the service which the entity is associated i.e. Army, Navy, Air Force.
- `sourceData` captures information about where data for the entity was sourced from. This is so the original source can be queried if required and lends authority to the data. A confidence level is applied to the data based on the accuracy of the source.
- `security` defines the security classification of the data.
- `repositoryID` identifies the repository in which the entity was created. It is used to assist data management between multiple service instances. More information about how this is used can be found in Section 9.6.

- `lazy` is used when fetching to indicate if an entity is a cut down, lazily loaded version of the full entity. For more information on this see Section 6.4.

This common `DataType` class is extended by more specific types which are then used to build up the ORBAT structures.

`UnitType` is the main building block of the ODM and is used to represent 'units' or 'sub-elements', henceforth referred to as units. Each unit can be allocated references from the Aide Memoire Data Service (AMDS) to indicate the holdings such as persons, equipment and containers of the unit. Some fields are specific to units and are defined in `UnitDataType`. These are:

- `noAmObjRef` which records the number of Aide Memoire references the unit has. These will be described more shortly. The purpose of this field is to assist with lazy loading as it allows for partially completed objects to be quickly fetched without retrieving all of the aide memoire references, but still being useful by being able to query the number of references the unit has.
- `position` represents a unique geospatial position where the unit is located. This is generally not stored in the service, but is provided for when it is required to record the disposition (i.e. physical arrangement or positioning) of an ORBAT.
- `type` is used to differentiate between units which are Capability Bricks (type = *BRICK*) or units which refer to the force in being (type = *INSTANCE*).
- `maintenanceType` is used to describe the type of maintenance which a unit is capable of providing. Valid options are *Integral*, *Close*, or *General*.
- `wuc` is used to define the default Weapon User Category for the unit.

The ODM allows units to loosely reference Aide Memoire (AM) objects such as persons or equipment. Inside the `UnitType` there exists a field called `amObjRefs` which is a list of `UnitTypeAmObjRef` where each object represents a reference to an AM object. Each `UnitTypeAmObjRef` contains a number of fields:

- `type` indicates the AM type which the object references, i.e. Equipment, Person, Container, Supply Item or Facility.
- `amid` is the unique identifier for the AM object so that it can be found in the AMDS.
- `name` is the name of the AM object at the time which the object was referenced.
- `version` is the version of the AM object at the time it was assigned to the unit entity. This can be used to see if the AM object has been updated in the AMDS since being assigned to the unit.
- `maximum` is used to define the maximum quantity of an AM object which has been assigned to the unit. When describing a Unit Entitlement (UE) this is the Operational Level of Capability (OLOC)[1] entitlement. This is the default quantity which should be used if no others are specified.
- `minimum` is used to define the minimum quantity of an AM object which has been assigned to the unit. When describing a UE this is the Minimum Level of

1 OLOC is the task-specific level of capability required by a force to execute its role in an operation at an acceptable level of risk. In other words it is the equipment requirement for the unit at OLOC.

Capability (MLOC)[2] entitlement. This is not populated for Capability Bricks which only use the maximum.

- `required` is used to define the Full Time Entitlement (FTE)[3] when describing a UE.
- `amRef` is a collection of `AmObjMRefType` objects which allows for users to describe other capability types/quantities for the AM reference. When describing a UE this will contain the Loan Entitlement (LE)[4]. Users are able to append other capability types if required.
- `wuc` specifies the Weapon User Category for the AM object.
- `prescribed` indicates if the unit has additional specialist capability for the piece of equipment and can therefore perform all levels of maintenance on that piece of equipment, i.e. Close (1st Line)/Integral (2nd Line)/General (3rd/4th Line).

The `AmObjMRefType` is used to define the quantity of an AM reference, i.e. the amount of the AM object which is assigned to the unit. The `capabilityType` is used when multiple quantities might be assigned to the unit for different situations and `multiplicity` defines the quantity.

As an example there may exist an A COY unit (`UnitType`) which contains two references (`UnitTypeAmObjRef`), a reference to a person and a rifle. When describing UEs all of the fields in the `UnitTypeAmObjRef` objects need to be defined. When describing capability bricks only the `maximum` quantity needs to be defined as the other quantities (`minimum`, `required` and the `amref` collection) are UE specific and are optional for capability bricks.

The `capabilityType` value is set depending on the AM capability being defined. For `maximum` it would be set to OLOC, for `minimum` it would be set to MLOC, for `required` it would be set to FTE and when putting in the `amRef` collection it will be LE. `UnitTypeAmObjRef` has been designed such that it can be extended in the future by allowing for a list of quantities in case there are extra capability types created.

Figure 6 gives an example in Java of assigning a Rifle to a UE unit.

2 MLOC is the lowest level of capability from which a force element can achieve its OLOC within readiness notice, and it encompasses the maintenance of core skills, safety and professional standards. Readiness Notice is the specified amount of time in which a force is to complete its workup from MLOC to OLOC.

3 The FTE is the level of equipment necessary to enable a unit to conduct routine training and administratve activities. A units FTE is its authority to hold Principal Items of equipment on permanent issue. The FTE may be less than the MLOC entitilement, with the shortfall being made availbale through an entitlement to loan equipment.

4 Loan pools provide equipment for simultaneous training at authorised scales for a balanced proportion of Australian Regular Army training establishments, the General Reserves, integrated units and Cadet units authorised to draw from the pool. The entitlement for units to draw this equipment is termed the Loan Entitlement (LE). LE is the unit's entitlement to hold equipment on a temporary basis.

```
UnitType unit = getUnit();

UnitTypeAmObjRef amRef = new UnitTypeAmObjRef();
amRef.setAmid("492771479923672-1178510364941");
amRef.setType(AmObjectTypeEnum.EQUIPMENT);
amRef.setName("F88 Steyr^");
amRef.setVersion(18);

AmObjMRefType am = new AmObjMRefType();

// OLOC
am.setCapabilityType("OLOC");
am.setMultiplicity(6.0);
amRef.setMaximum(am);

// MLOC
am = new AmObjMRefType();
am.setCapabilityType("MLOC");
am.setMultiplicity(6.0);
amRef.setMinimum(am);

// FTE
am = new AmObjMRefType();
am.setCapabilityType("FTE");
am.setMultiplicity(6.0);
amRef.setRequired(am);

// LE
am = new AmObjMRefType();
am.setCapabilityType("LE");
am.setMultiplicity(0.0);
amRef.getAmRef().add(am);

unit.getAmObjRefs().add(amRef);
```

Figure 6: Java code showing the creation of a valid Unit Entitlement Aide Memoire reference and assigning it to a unit

The other main component of the ODM is `ORBATType` which is used to represent 'ORBATs' of entire forces or individual 'Force Elements' that compose a force. This acts as a container for units and the relationships between them. Each ORBAT inherits the common fields from `DataType` but also has some ORBAT specific fields. `ORBATDataType` has the fields:

- `nationality` defines the nationality of the ORBAT. It uses country codes from the FIPS Pub 10 series. This is used in the MIL-STD-2525B and is also a part of the `symbol2525B` code.
- `noUnits` indicates the number of units associated with the ORBAT. This is used in lazy loading so when the full graph isn't fetched from the service the number of units can still be found.

- `noLinks` indicates the number of links associated with the ORBAT. This is used when lazy loading so the number of relationship links can still be found.
- `structureType` defines the type of ORBAT that is being represented. The service supports the storage and retrieval of different types of ORBATs which follow different processing rules such as *Capability Bricks* and *Unit Entitlements*. The valid values for this are `OU_BRICK` for a capability brick made up of units (ORBAT of Units), `OO_BRICK` for a capability brick made up of ORBATs (ORBAT of ORBATS), `UE` for a Unit Entitlement ORBAT containing units, `OO` for a Unit Entitlement ORBAT of ORBATs and `OPO` for a Defence Preparedness Requirements (previously Operational Planning Objective) shopping list. Note that the terms OPO and DPR will be used interchangeably in this report.

4.1 Relationships

`ORBATType` has a list of units and a list of links which are used to define the relationships between those units. Relationships are defined by using `RelType` objects which have a number of fields:

- `name` is used to record a name/label for the relationship which may be shown to the user.
- `type` defines the relationship type such as command, support, maintenance etc.
- `parentOrbatIid`, `parentOrbatVid`, `childOrbatIid`, `childOrbatVid` are used to define a link between ORBATs. The fields which are populated determine how the link is processed.
- `ParentUnitIid`, `parentUnitVid`, `childUnitIid`, `childUnitVid` are used to define a link between units within an ORBAT. How the fields are populated determine how the link is processed.
- `cardMin` is the minimum cardinality of the relationship which has a default value of 0.
- `cardMax` is the maximum cardinality of the relationship and has a default value of 1.

As mentioned, the combination of fields populated determines how a link is processed. Table 2 shows the valid combinations of fields for ORBATs of different structure type.

Relationships are defined between units or ORBATs but not both, that is, the ORBAT service does not support defining a relationship between an ORBAT and a unit. Relationships between ORBATs can only be defined in ORBATs with `structureType` `OO`, `OU_BRICK` or `OPO`.

Table 2: Matrix showing the valid combination of RelType fields based on the structureType of the containing ORBAT. A 'Yes' indicates that the field can be populated. A '-' indicates it shouldn't be populated.

StructureType / RelType	pOIid	cOIid	pOVid	cOVid	pUIid	cUIid	pUVid	cUVid
UE	-	-	-	-	Yes	Yes	Yes	Yes
OO	Yes	Yes	Yes	Yes	-	-	-	-
OU_BRICK	-	-	-	-	Yes	Yes	Yes	Yes
OO_BRICK	Yes	Yes	Yes	Yes	-	-	-	-
OPO (Unit)	-	-	-	-	Yes	-	Yes	-
OPO (ORBAT)	Yes	-	Yes	-	-	-	-	-

4.1.1 Static Link vs Dynamic Link

As mentioned in Section 5 there are two different ways to construct links which affects the fetching strategy that is used. These are static and dynamic links.

To define a static link relationship between two entities the parent and child iid are the only fields that are populated on the `RelType` object. To define a dynamic link between two entities then both the parent `iid` and `vid` must be defined, as well as the child `iid` and `vid`.

The reason the `iid` is defined on a dynamic link as well as the `vid` is because the `iid` fields on a `RelType` object are actually an XML Schema IDREF and so having them specified can increase performance and simplify searching the entity graph.

4.1.2 Relationships Between ORBATs

ORBATs with `structureType` `OO`, or `OU_BRICK` are referred to as ORBAT of ORBATs, or ORBOs, for want of a better term. These types of ORBAT can only contain other ORBATs and can not contain any units. They are used to define the associations and links between different ORBATs. In the case that a command link is defined between two ORBATs it is assumed that the root unit inside the parent ORBAT (HQ) commands the root unit within the child ORBAT. The reason that this is assumed is that it is not technologically possible for the ORBATDS to define unit to unit links across ORBATs. Unit to unit links contain an IDREF in the parent/child `iid` fields. When the link is marshalled, if the linked unit is not inside the ORBAT being sent in the request then the link is

persisted with the iid being *null*, thus creating an invalid link and preventing the ORBAT from being persisted.

To define an ORBAT relationship within an ORBO you need to define a `RelType` with `type` set as `ASSOCIATION` between the container ORBO and the associated ORBATs and store it in the links list of the ORBO. This will have the parent set as the ORBO and the child set as the associated ORBAT. In addition, for ORBATs of type *OO*, or *OU_BRICK* (but not *OPO*, they can only contain `ASSOCIATION` links) it is also possible to define other relationship such as `COMMAND` relationships. These are constructed by creating a `RelType` object with type `COMMAND` with the parent set to the commanding ORBAT and the child set as the subordinate ORBAT.

Figure 7 shows how the links for an ORBAT of ORBATs representing 1 Brigade would be constructed:

```
ASSOCIATION link: 1 BDE == HQ 1 BDE
ASSOCIATION link: 1 BDE == 1 ARMD REGT
ASSOCIATION link: 1 BDE == 1 CER
ASSOCIATION link: 1 BDE == 1 CSR
ASSOCIATION link: 1 BDE == 2 CAV REGT
ASSOCIATION link: 1 BDE == 5 RAR
ASSOCIATION link: 1 BDE == 7 RAR
ASSOCIATION link: 1 BDE == 1 CSSB
ASSOCIATION link: 1 BDE == 8/12 MDM REGT
COMMAND link: HQ 1 BDE ==> 1 ARMD REGT
COMMAND link: HQ 1 BDE ==> 1 CER
COMMAND link: HQ 1 BDE ==> 1 CSR
COMMAND link: HQ 1 BDE ==> 2 CAV REGT
COMMAND link: HQ 1 BDE ==> 5 RAR
COMMAND link: HQ 1 BDE ==> 7 RAR
COMMAND link: HQ 1 BDE ==> 1 CSSB
COMMAND link: HQ 1 BDE ==> 8/12 MDM REGT
```

Figure 7: An example of how links are constructed within a valid ORBAT of ORBATs. '==' indicates a non-directional link as ASSOCIATION links have no direction. '==>' indicate a directional link as the parent commands the child.

The ORBAT to ORBAT `RelType` objects stored in ORBOs must be structured as follows:

- To define a **dynamic** relationship between two ORBATs the Parent ORBAT Version Identifier, *vid* (which refers to a specific version of an ORBAT), Child ORBAT Version Identifier, *vid*, the Parent ORBAT Instance Identifier, *iid*, and the Child ORBAT Instance Identifier, *iid*, need to be exclusively defined. That is, they must all have values while the other fields must remain null.
- To define a **static** relationship between two ORBATs both the Parent ORBAT Instance Identifier, *iid* (which refers to a specific ORBAT instance) and Child ORBAT Instance Identifier, *iid* need to be exclusively defined.
- For all ORBAT-to-ORBAT links only *ASSOCIATION* or *COMMAND* links can be defined.

For capability brick ORBOs, of structure type `OO_BRICK`, a cardinality can also be defined on the link. This allows for defining relationships such as "ORBAT A commands 3 of ORBAT B". To define the cardinality the `cardMax` field is set on the `RelType` object.

4.1.3 Relationships Between Units

Relationships between units within an ORBAT can only be defined within ORBATs of type `UE` or `OU_BRICK` as these are the ORBATs which contain units. The data model supports defining different types of relationships such as support or maintenance, but only command relationships can be persisted in the service.

To define a relationship between units within an ORBAT a `RelType` object must be structured as follows and stored within the ORBAT:

- To define a **dynamic** relationship between two units the Parent Unit Version Identifier, *vid* (which refers to a specific version of a Unit), Child Unit Version Identifier, *vid*, Parent Unit Instance Identifier, *iid*, and Child Unit Instance Identifier, *iid*, need to be exclusively defined. That is, they must all have values while the other fields must remain null.
- To define a **static** relationship between two units both the Parent Unit Instance Identifier, *iid* (which refers to a specific Unit instance) and Child Unit Instance Identifier, *iid* need to be exclusively defined.

For capability bricks of structure type `OU_BRICK`, a cardinality can also be defined on this link. To define the cardinality the `cardMax` field is set on the `RelType` object.

4.1.4 Relationships in a DPR

A DPR ORBAT is slightly different than the other structure types. The DPR can be thought of as a shopping list of different ORBAT and unit entities. It doesn't contain any command structure between the entities. Because of this only *ASSOCIATION* links can be defined. DPR ORBATs also allow cardinality to be defined for the entities it contains. Both the minimum and maximum cardinality can be defined to express relationships such as 'This ORBAT contains between 3 and 5 of this unit'. The maximum cardinality must not be less than the minimum cardinality and both default to 1.

- To define a **dynamic** relationship to an entity the parent (and only the parent) instance identifier, `iid`, and version identifier, `vid`, need to be defined.
- To define a **static** relationship to an entity only the parent instance identifier, `iid`, needs to be defined.

In both cases, the identifiers used on the `RelType` object are for the ORBAT or unit being included in the DPR. Every entity should have an associated relationship defined in the DPR.

4.2 Type Restrictions

When developing an ORBAT there are some restrictions on the type of units and ORBATs which can be included based on the structure type of the ORBAT being constructed. These restrictions are listed in Table 3 below.

Table 3: Table shows the containment type restrictions. An 'X' indicates that type of entity can't be contained in an ORBAT with the specified structure type.

Structure Type:	UE	OO	OU_BRICK	OO_BRICK	OPO
Unit Type	INSTANCE	X	BRICK	X	BRICK
ORBAT Type	X	UE, OO	X	OU_BRICK, OO_BRICK	OU_BRICK, OO_BRICK

Cells which have a cross in them indicate that entities with the given type can not be included in the ORBAT. These restrictions are to prevent the mixing of Unit Entitlement data, which is based on real forces, and capability brick data which don't have a defined real force.

4.3 Capability Brick Construction

A capability brick is a generic force structure which characterises a kind of capability and are used to build up a force structure for a contingency. There can be both ORBAT and unit capability bricks, for example a Ready Combat Team (RCT) would be an ORBAT capability brick and a Head Quarters (HQ) would be a unit capability brick.

The construction of capability bricks is similar to the construction of UEs with a couple of differences. Firstly, the structure type for capability brick ORBATs must be `OO_BRICK` or `OU_BRICK`, and the units must be of type `BRICK`. The other difference is that capability bricks can contain cardinality on the *COMMAND* links by setting the `cardMax` field on the `RelType` object.

ORBATs should be modelled in a particular way to increase the re-usability of components. Figure 8 shows a diagram of what a capability brick we want to model in the service might look like while Figure 9 shows how this capability brick would be modelled using the ODM.

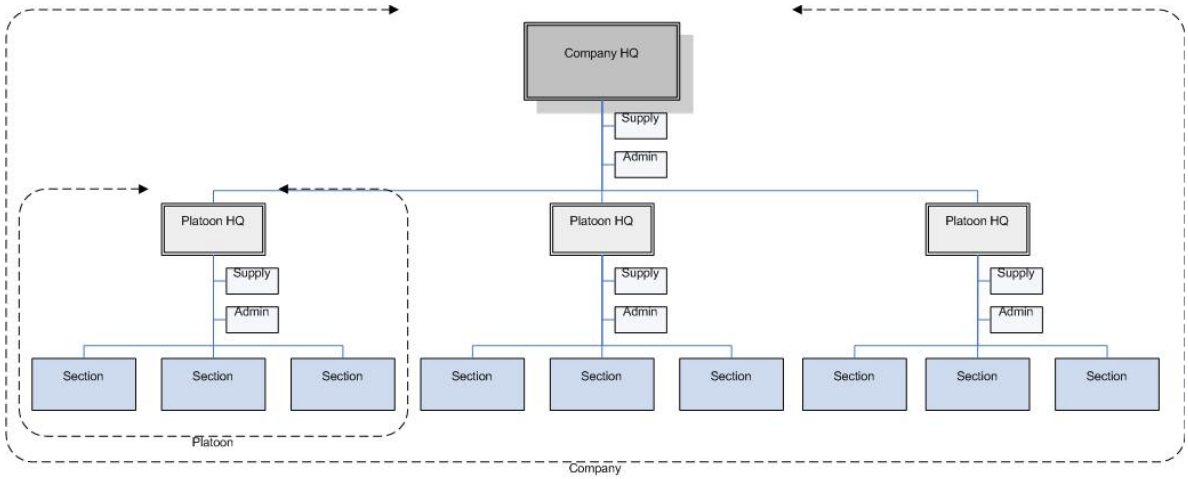

Figure 8: A wire diagram of the ORBAT for a generic company with three generic platoons, each with three generic sections

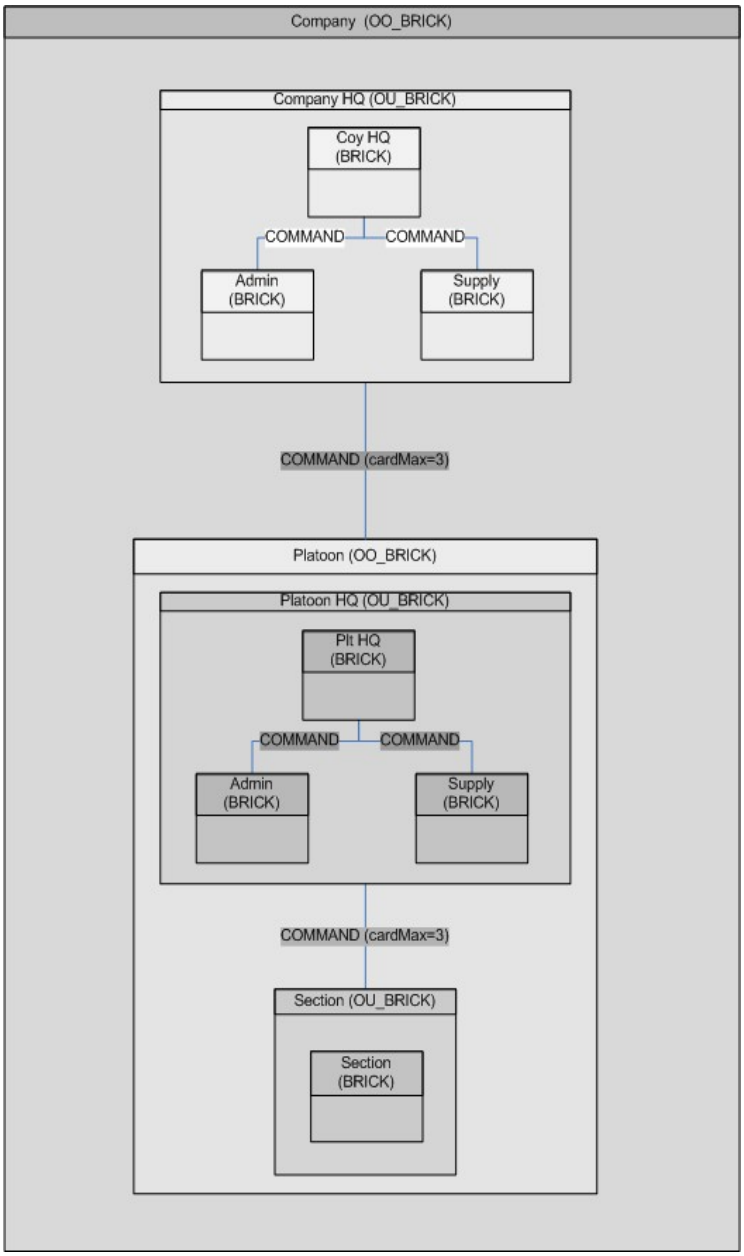

Figure 9: A representation of how the ORBAT in Figure 8 would be represented in the ORBATDS Data Model

4.4 Mandatory Fields

There are a number of mandatory fields which need to be populated in order for the ORBATDS to accept the entity as valid and store it in the service. Some of the fields are mandatory to support the functionality described later in the report and/or to support the functionality of external tools, particularly ViPA. The set of mandatory fields depends on the structure type of the entity, as different types are used in different use cases and have different business rules applied.

There are some mandatory fields which are common across all types of entities, these are:

- Instance identifier (iid)
- Version identifier (vid)
- Start time (timeType.start)
- Structure type (for ORBATs) or type (for units)
- Primary capability
- Echelon
- Battle dimension
- Affiliation
- Name
- Formal name
- Service

In addition, capability brick ORBATs and Units are required to have role populated.

4.5 Symbology

The ORBATDS implements US Department of Defense Interface Standard MIL-STD-2525B with Change 2 [31] for symbology since this is the standard which Australian symbology is based on, as discussed in Section 1.1. This allows ORBATs to be displayed on different user interfaces such as in Figure 10.

The standard provides a standardised, structured set of graphical symbols for the display of information in command and control (C2) systems and applications. In joint military operations, it is imperative to have a common language clearly understood among all users. Graphical representation of units are observed and readily understood faster than text alone [31].

The ODM stores the 2525B 15-character alphanumeric SIDC which provides the information necessary for a system to transmit and display a tactical symbol and its modifier fields. The ODM also has separate fields for storing the information which is encoded in the symbol such as primary capability, echelon, battle dimension, affiliation and nationality so this information is known in systems which don't implement MIL-STD-2525B. The values of these fields should be based on the standard.

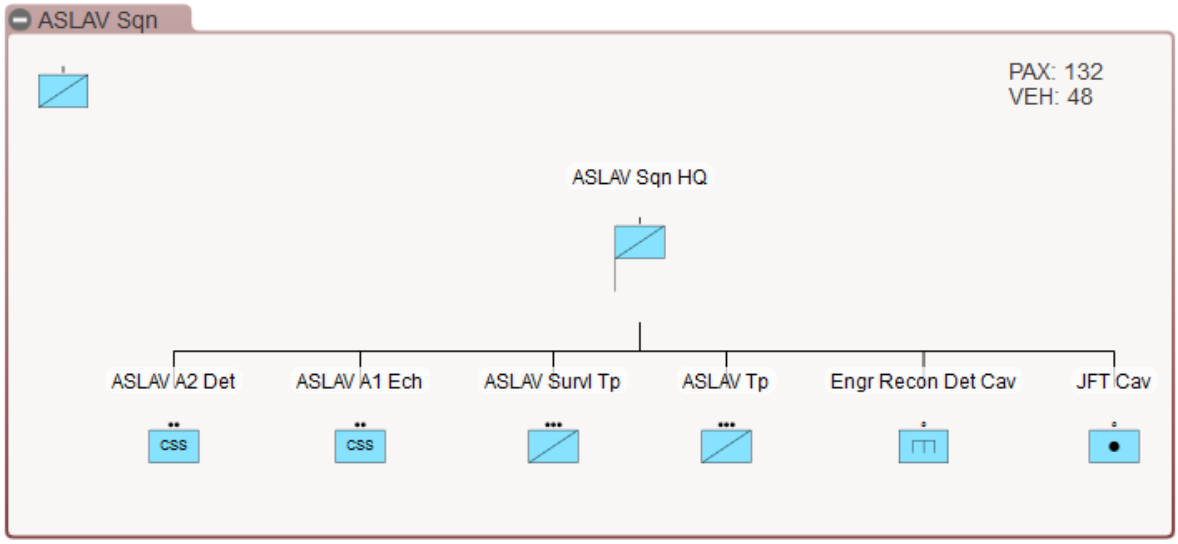

Figure 10: An example display of an ORBAT using MIL-STD-2525B symbology. This diagram is generated by the ORBAT Builder web application currently being prototyped by DST Group. More information can be found about this in Section 11.6.

The ViPA Data Manager who is responsible for the consistent use of descriptors and symbology based on US MIL-STD-2525B estimates that the standard covers only 75% of ADF units.

5. Temporal Modelling and Entity Versioning

The ORBAT Data service provides a novel solution to capturing and representing the evolution of force structures over time. This is to meet requirement 7 in Section 3 for enabling the service to represent unit rotations and changes to unit capability. This was achieved by providing each entity with a time line along which different versions (major changes) and revisions (minor changes within a version) are managed. The term 'entity' is used here when addressing both units and ORBATs.

When users update an entity they may choose to create a new version which has its own time line defined starting from the date at which those changes apply to the real world object represented by that entity. For example, a new entity representation might be created with a time period starting on the 1st of January 2020. Later the user decides to make a new version of the same entity to record a unit rotation occurring on the 1st of February 2021. The initial version will now have a time line defined from the 1st of January 2020 to the 1st of February 2021 and a second version with its own timeline starting on the 1st of February 2021.

In addition to major versions, the service also maintains the history of all minor changes or corrections that do not impact on unit capability and include changes like correcting a typographic error in the unit's description or augmenting a model by adding missing

information. The history is formed by capturing these different 'revisions' of the entities to meet requirement 14 so as to ensure that changes made to an entity are not lost.

A two dimensional revision control system was designed to ensure that all changes made to an entity are retained. Revision Control Systems (RCS) offer many benefits that have led to them being widely adopted across a number of domains. The software engineering field being the most prominent adopter, has been using RCS since the early 1970s [12, 13] to manage changes to software, particularly when being made by teams of developers. The ORBATDS implements a RCS for entities stored in the service. The main benefits of having a RCS are maintaining historical records and the ability to rollback changes. This promotes data sharing which encourages collaboration and thus helps to improve data quality [14, 15, 16]. As all changes are reversible the data entry risk is reduced, encouraging subject matter experts to contribute without fear of losing any of their data, thereby wasting their time and effort [17].

This system allows client applications which require a specific version and/or revision of an entity to query and fetch that specific represntation for its purposes, but also to be able to view past versions and/or revisions and determine how entities have changed over time.

5.1 Temporal Design

When creating an ORBAT or a unit, each change to an entity will either create a new version or revision of the entity. A version is the representation of an entity for a particular period of time, while a revision describes a minor change to a version during that time period. A new version of an entity is intended to be created when an entity undergoes a significant change to its capability. This might occur when an organisation is restructured or equipment is upgraded providing a significantly enhanced capability. The new version would become effective from whenever the change is implemented.

For example, the F-111 strike fighter is a supersonic long-range tactical strike aircraft that was in service since the early 1970's and in late 2010 was replaced with the F/A-18F Super Hornet, giving the military a significantly upgraded air combat capability. In this case the asset was upgraded resulting in a major change to the air combat capability of No. 82 Wing. Consequently, the temporal model of No. 82 Wing would have two versions; the first has the F-111 equipment and spans from the early 1970's up until late 2010. The second version has the F/A-18F equipment and spans from late 2010 to some as yet unknown point in the future.

To more formally define the temporal modelling of the system; every entity in a model can contain m versions where $m >= 1$. Each version has a distinct time line defined by a start date and an end date. The start date is defined when an entity is initially created and all subsequent versions of that entity must exist after this initial start date. The end date on the other hand is initially undefined, indicating that the version currently applies and has an unknown end date. When a new version is created the start date of the new version

must follow the end date of the previous version. Therefore the end date of the previous version is set to be the start date of the new version, thus creating a continuous time line for each entity where no versions overlap.

For each version of an entity there also exists *n* revisions where n >= 1. Each revision of a version applies for the whole time period of that version, usually because it's a correction to that version. When a version is created it is said to be the first revision of that version. As subsequent revisions are made, the revision which has the latest changes denotes the *head* revision and supersedes all previous revisions. The determining factor used by the ORBATDS when deciding whether a change should result in a new version or revision is if the start date of an entity has been modified.

Figure 11 illustrates the changes to an entity over time. Here you can see an entity which has evolved through three separate versions, each of which contain multiple revisions. The *head* revision of each version has been highlighted by a striped pattern. Note that the end date of version 3 (V3) is undefined, indicating that it is the latest version.

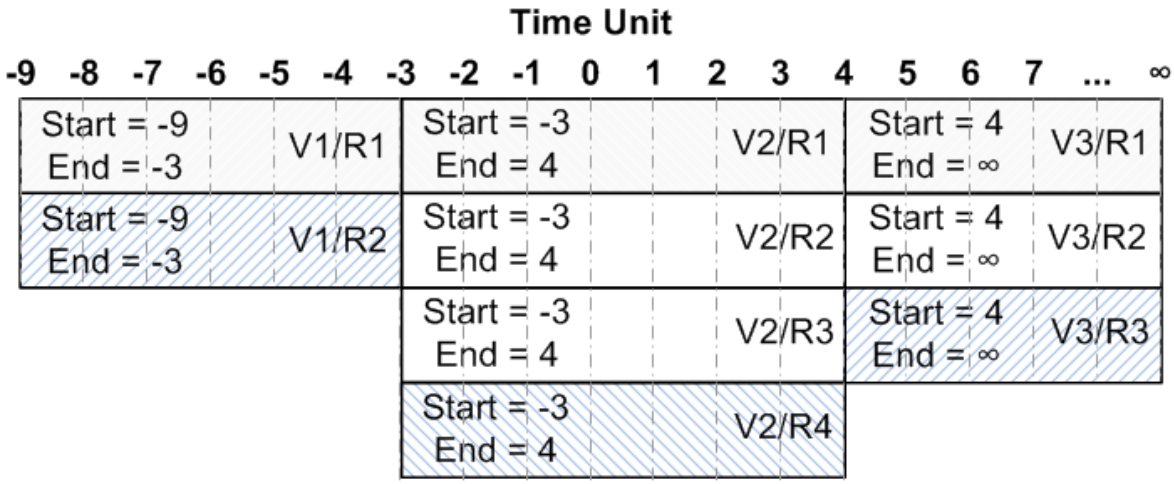

Figure 11: The evolution of an entity over time. There are three versions of the entity, each defined for different periods along the horizontal axis. Each version has multiple revisions shown along the vertical axis. The striped areas indicate the head revision of each version. The time unit of 0 represents the present time, with negative units representing the past and positive units representing the future.

The latest version of an entity will always have its end date defined as *null* which is used to indicate infinity. Every version of an entity will have its end date set to the start date of the next version, ensuring that each entity has a continuous life cycle of versions with no overlaps.

5.2 Temporal Linking

To support working with this temporal model for units and ORBATs, two different linking techniques have been implemented which are known as *dynamic linking* and *static linking*.

These linking techniques determine the behaviour of the fetching strategies, which retrieve the particular revision needed by the user application. A static link references a particular instance of an entity, that is, a particular revision of a particular version. Every time an ORBAT containing a static link is retrieved from the service, the exact instance of the linked entity will be returned. This makes it possible to create and recreate the exact same snapshot of an entire ORBAT based on specific versions/revisions of entities.

In contrast, a dynamic link references a particular entity, but not which specific version/revision of that entity. A fetching strategy is then used when retrieving an ORBAT from the ORBATDS and the latest revision of the linked entity will be resolved and returned to the client application. However, the version that is retrieved will depend on the particular fetching strategy being used. Any changes which have been made to the linked entities will be reflected in the ORBAT returned from the service. The advantage of the *dynamic linking* technique is that it allows each entity to be managed independently. Any modifications made are automatically propagated throughout the models which are dependent on the entity. This linking technique also gives data consumers the guarantee that entities retrieved contain the most up-to-date information.

6. Fetching Strategies

When retrieving an ORBAT it is likely that the ORBAT and the entities within it have multiple versions (see Section 5). A number of fetching strategies are provided that use a set of rules to determine which version of each entity is included in a response to the client application. For example, when using an ORBAT in ViPA to support immediate planning the user will need the current version of that ORBAT, where as someone using ViPA to support deliberate planning for possible future contingencies will need the latest version of an ORBAT, which could be defined to be in the future according to the scenario being addressed. This section outlines the different fetching strategies that are available to client applications.

6.1 Temporal Fetching Strategies

There are three different fetching strategies that determine how entities which are dynamically linked are returned from the timeline. The first strategy is used to fetch the **current** version of an entity (i.e. the head revision of the current version for time unit 0), the next is used to fetch the **latest** version of an entity (i.e. the head revision of the version furthest into the future) and the final one is used to fetch a specific **instance** of an entity (i.e. the head revision of a specific version).

A simple example will be used to introduce the fetching strategy concept. This example has an ORBAT O1, which contains two units U1 and U2. There is a dynamic link between U1 and U2. Each entity contains multiple versions and revisions. Assume the **current** version of the ORBAT O1 is to be fetched. This is the recommended fetching strategy for

most use cases as it is designed to retrieve the version of each entity which applies to *now*. Figure 12 shows how this strategy looks.

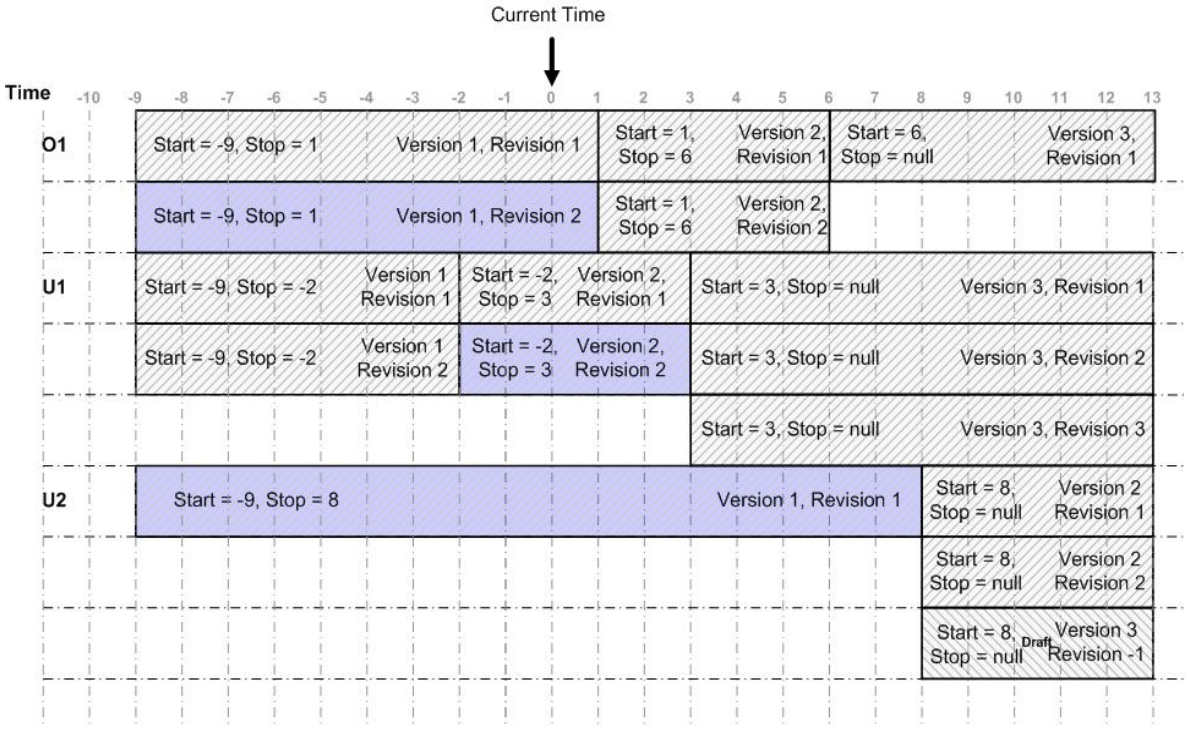

Figure 12: This figure illustrates the current fetching strategy for an ORBAT containing two units which are dynamically linked

The **current** strategy shown in Figure 12 uses the current time at Time = 0 to resolve each entity to the version which intersects this point of time. In this example it will return those highlighted in the figure: O1 (V1, R2), U1 (V2, R2) and U2 (V1, R1). If the current time lies at the border between two versions, the version to the right will be returned.

In the **latest** strategy the service will resolve entities to their latest version. This strategy is illustrated in Figure 13. In this example it will return the highlighted entities: O1 (V3, R1), U1 (V3, R3) and U2 (V2, R2). Note that U2 has a draft revision which has yet to be published. More information about drafts and the data management system can be found in Section 9.

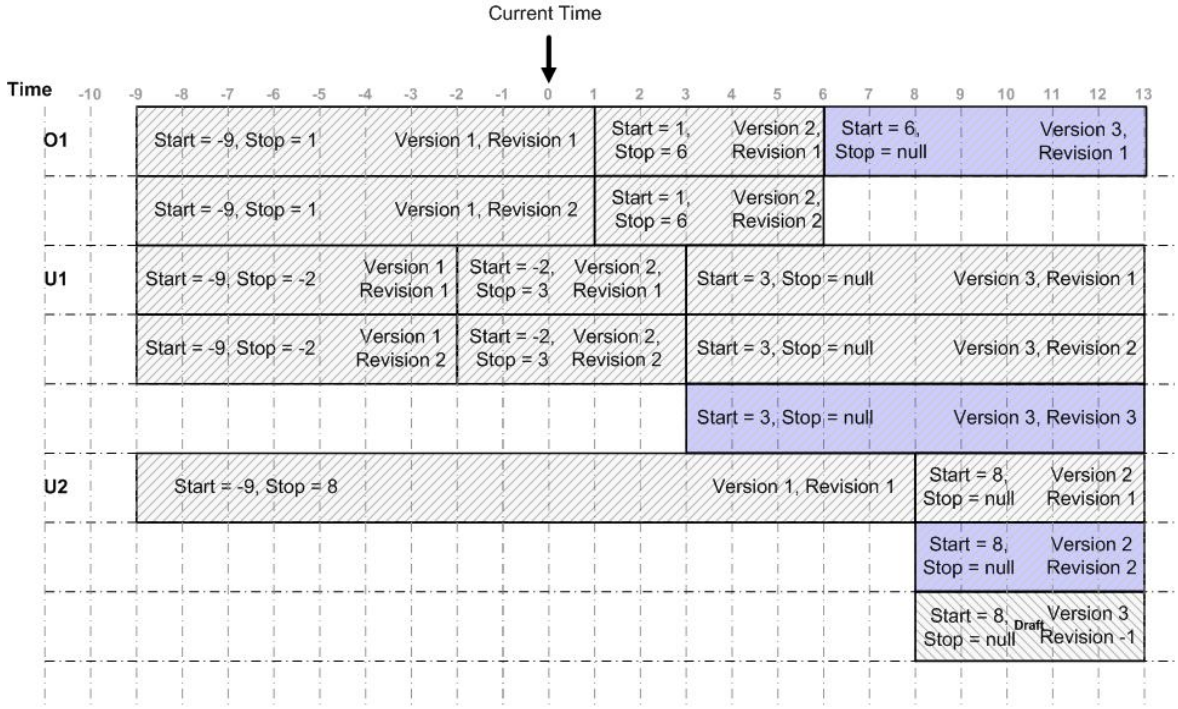

Figure 13: This figure illustrates the latest fetching strategy for an ORBAT containing two units which are dynamically linked

In the **Instance** strategy the service will return a specific version and revision of an entity as specified by its unique identifier, thus making it possible to get old revisions of entities. When resolving dynamic links the service will fetch the latest version which exists within the time period of the ORBAT entity being retrieved. Figures 14, 15 and 16 illustrate this instance fetching strategy. In Figure 14 the client has requested O1 (V1, R1). When resolving the dynamic links it will return the latest revision of the latest version of U1 and U2 which intersect the end date of the ORBAT. In this case U1 (V2, R2) and U2 (V1, R1).

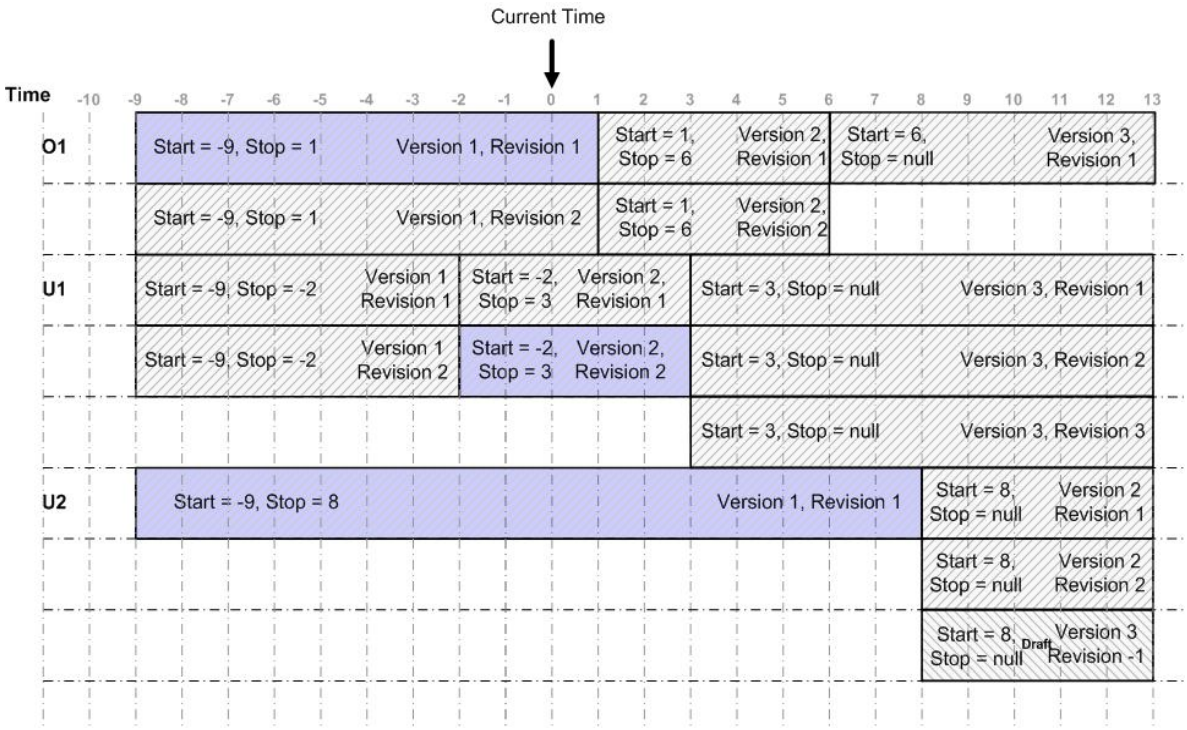

Figure 14: This figure shows which instances of each entity will be retrieved when using the instance fetching strategy to retrieve version 1, revision 1 of ORBAT O1 that contains units U1 and U2 which are dynamically linked

In Figure 15 the client has requested O1 (V2, R2). The dynamic links are again resolved to the latest most versions which exist within the ORBAT time frame which are U1 (V3, R3) and U2 (V1, R1).

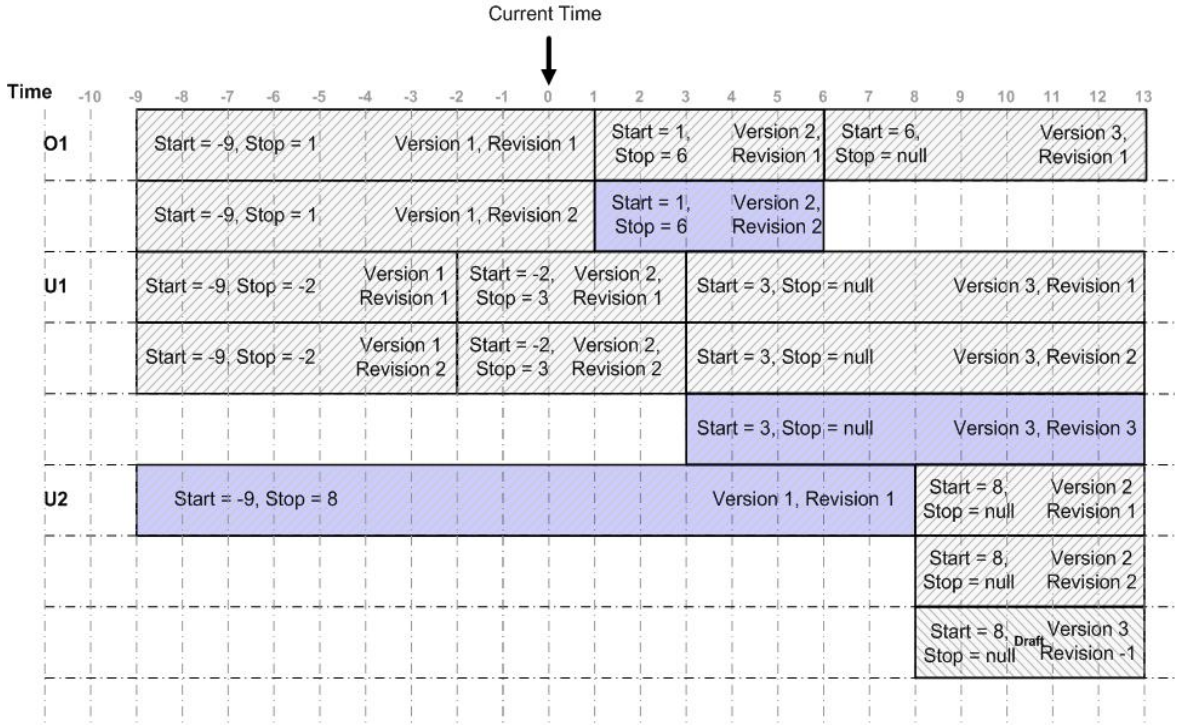

Figure 15: This figure shows which instances of each entity will be retrieved when using the instance fetching strategy to retrieve version 2, revision 2 of ORBAT O1 that contains units U1 and U2 which are dynamically linked

Finally in Figure 16, the client has requested O1 (V3, R1). In this case the ORBAT does not have an end date (Stop = null) which implies it extends forward to infinity thus the latest versions of the dynamically linked units are returned, U1 (V3, R3) and U2 (V2, R2).

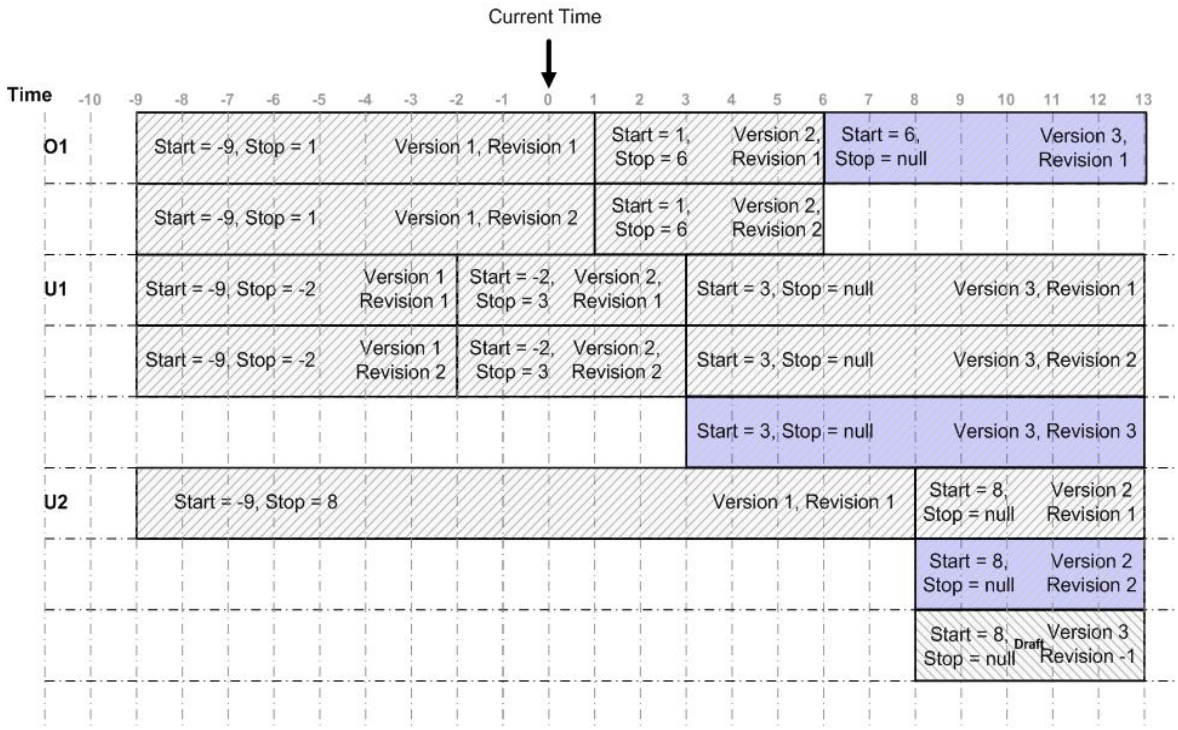

Figure 16: This figure shows the instance fetching strategy for an ORBAT containing two units which are dynamically linked

Another example will be given using an entity command hierarchy shown in Figure 17 with three entities where entity E1 commands both entities E2 and E3.

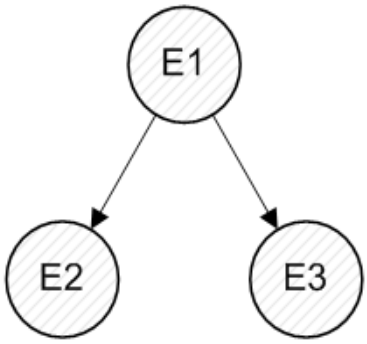

Figure 17: An entity command hierarchy

Figure 18 shows an example of how the entities shown in Figure 17 could be represented in the temporal model, visualising the versions and revisions of each entity.

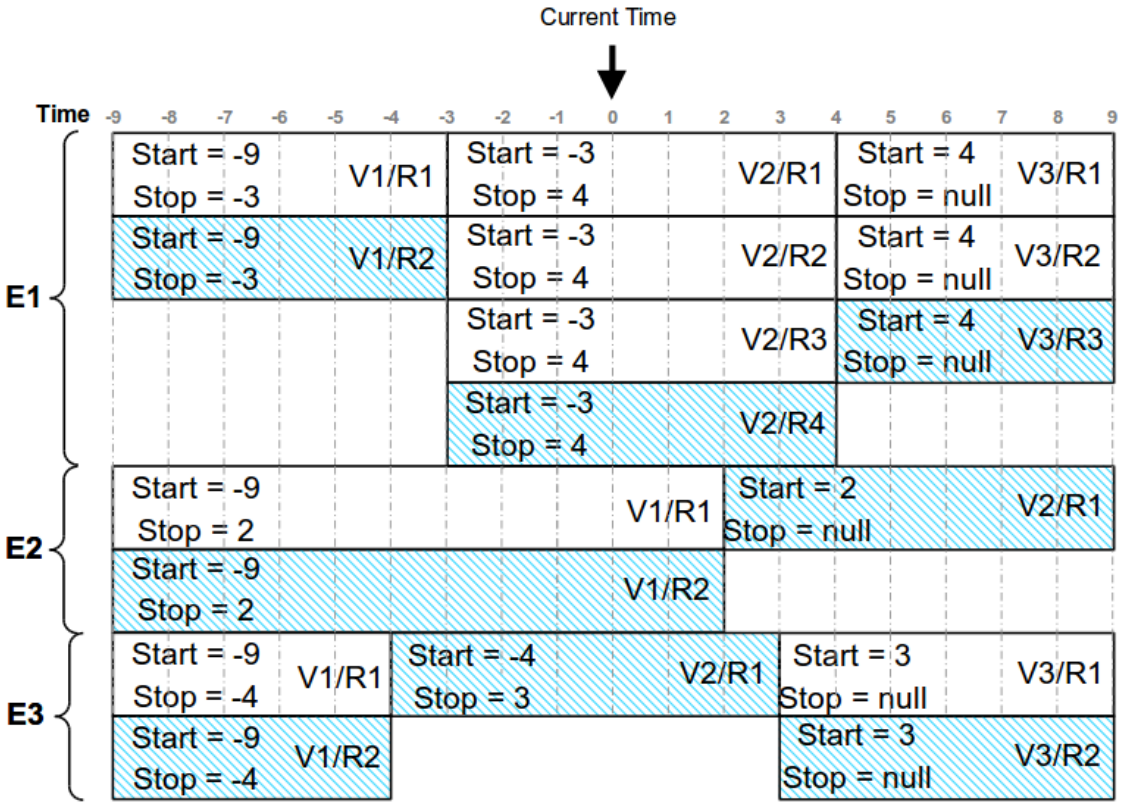

Figure 18: Three entities showing in light shading the head revision of each version

Figure 18 will be used to illustrate the following example, assuming that Time Unit 0 is the current point in time. Here the client retrieves the entity E1 including its dependent child entities, E2 and E3. When using the **current** retrieval strategy the *head* revision of the version of each entity that intersects the current time will be retrieved. This includes E1 (V2/R4), E2 (V1/R2) and E3 (V2/R1). If the **latest** retrieval strategy were used then the *head* revision of the latest version of each entity will be returned. This includes E1 (V3/R3), E2 (V2/R1) and E3 (V3/R2). Finally, if the **instance** retrieval strategy were used then the *head* revision of the version which intersects the end date of the chosen version of E1 will be retrieved. For example if retrieving E1 (V2/R3) then the *head* revision of the latest version of E2 and E3 which exists within the temporal bounds of E1 V2 will be retrieved, as shown by the highlighted boxes in Figure 19. The red vertical line shows the end date of E1 V2 and shows that the versions of E2 and E3 retrieved are those which intersect this date. As a side case, if the version of E1 had no defined end date then the *head* revision of the latest version of E2 and E3 will be retrieved.

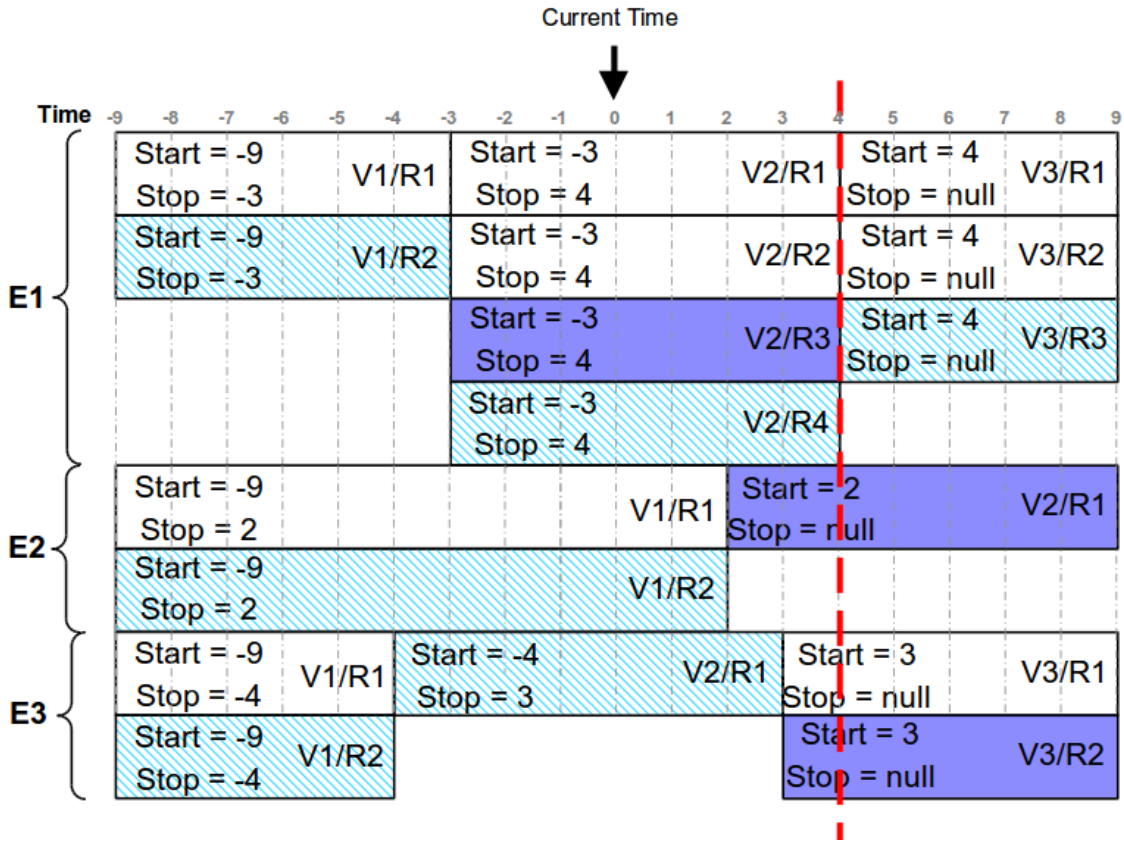

Figure 19: Illustration of instance entity fetching strategy. The solid dark areas show which entities are retrieved. The red vertical dashed line highlights the end date of E1 V2.

6.2 Timeline – Version Continuity

To ensure the correct operation of the system it is essential that each entity has a continuous timeline. If there was no continuity of versions then it would not be possible to guarantee that a linked entity can always be retrieved. Figure 20 illustrates an illegal example where the timeline for E2 is segmented. To illustrate, the earlier example of retrieving the instance E1 (V2/R3), shown in Figure 19, will be used. Using the **instance** retrieval strategy, no version of E2 can be retrieved as E2 does not exist within the temporal bounds of E1 V2.

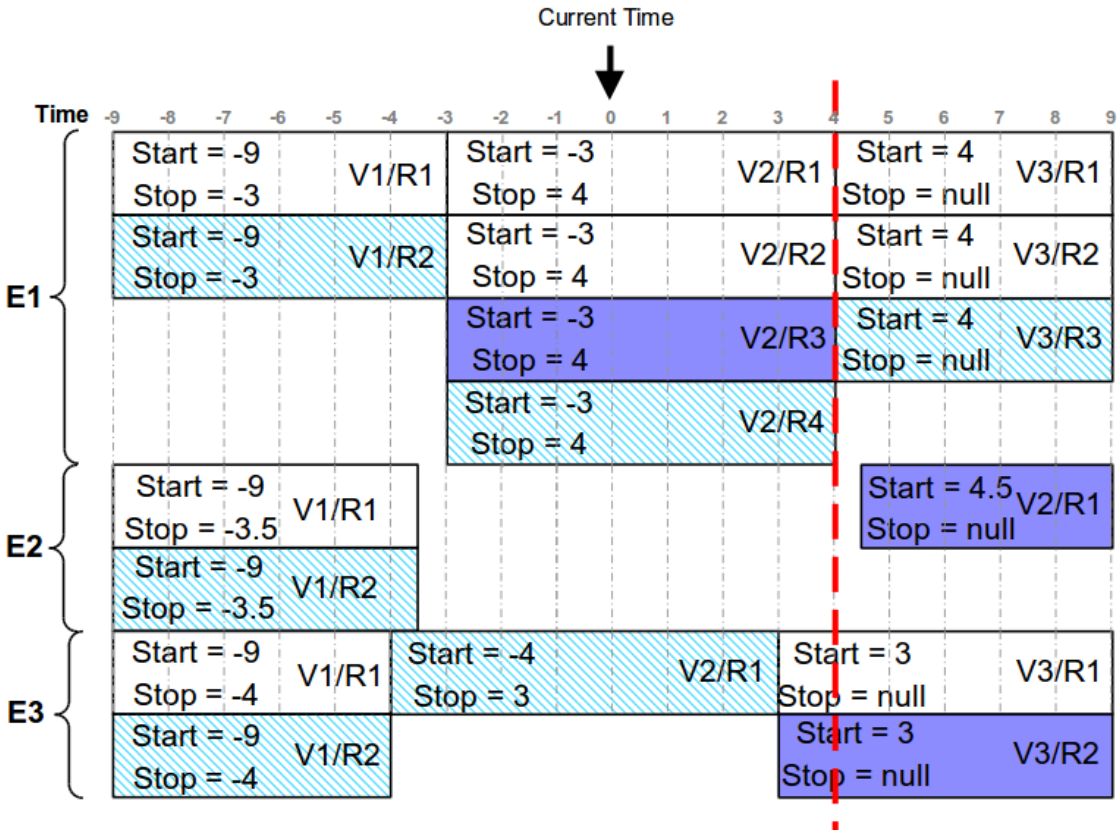

Figure 20: An example of temporal model discontinuity

6.3 Fetching Dependencies

When fetching an ORBAT it is useful to specify whether its dependencies should be retrieved or not. For example, a client application in one case might need to display the entire ORBAT hierarchy, while in another case only the aggregated ORBAT is needed. Furthermore, large ORBAT structures can be created in the service which can amount to a lot of data. It can be time consuming to build up the entire ORBAT structure from the database and return it to the client. Therefore, the ability to determine if dependencies should be fetched or not was introduced. When dependencies aren't fetched, only the parent ORBAT element that was requested is returned, thus significantly speeding up the operation.

6.4 Lazy Loading

Lazy loading is designed to increase the performance of fetching entities from the service and reduce the response time for clients. This acts as an intermediate approach which sits between the empty dependency fetch which only retrieves the entity specified and the full

dependency fetch which retrieves all entities and their dependencies. Lazy loading retrieves enough information to allow clients to process the ORBAT data without experiencing large latencies. Figure 21 highlights the three approaches that the service supports in order of increasing number of objects returned to the client.

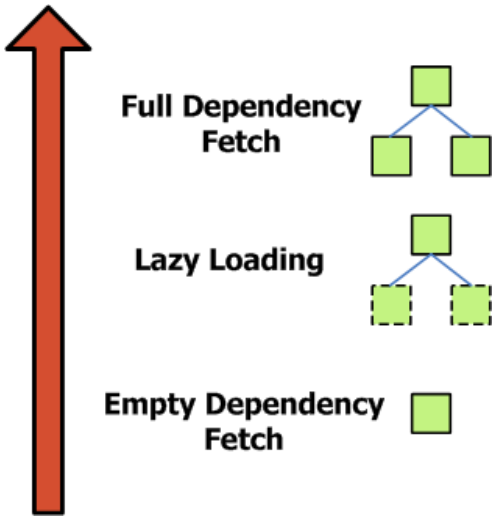

Figure 21: Diagram showing increasing number of objects for different approaches

These various in fetching approaches cause different data objects to be returned to the client as explained below:

- Concrete ORBAT: These ORBAT entities contain all data fields including both lists of links and units. These contain all the data from the persisted entity.
- Half-lazy ORBAT: These ORBAT entities are completely populated as for Concrete ORBATs, except their dependent entities are lazily loaded.
- Lazy ORBAT: These ORBAT entities are lazily loaded and do not contain the lists of links or units.
- Concrete Unit: These unit entities contain all data fields and a complete list of aide memoire references. Recall from Section 4, that each unit can be allocated references from the AMDS to indicate the holdings such as persons, equipment and containers.
- Lazy Unit: These unit entities are lazily loaded and do not contain any aide memoire references.

The returned data structure indicates whether it has been lazily loaded or not. When lazily fetching ORBATs, the ORBAT being requested will be half-lazy and its dependencies will be lazy.

The following diagrams illustrate the structures populated when using lazy loading and how they differ based on the type of entity. The top tree of each figure gives the persisted view of the entity (i.e. what is in the ORBATDS database) and then shows the lazily loaded version of that entity which is returned to the client.

6.4.1 ORBAT of Units

An ORBAT of Units describes an ORBAT which contains units in a command hierarchy. When lazily loading a structure of this type the ORBAT will be loaded, but the units it contains will be lazy (i.e. no aide memoire references). A subsequent call can be made to the service to fetch those missing references. This case is illustrated in Figure 22.

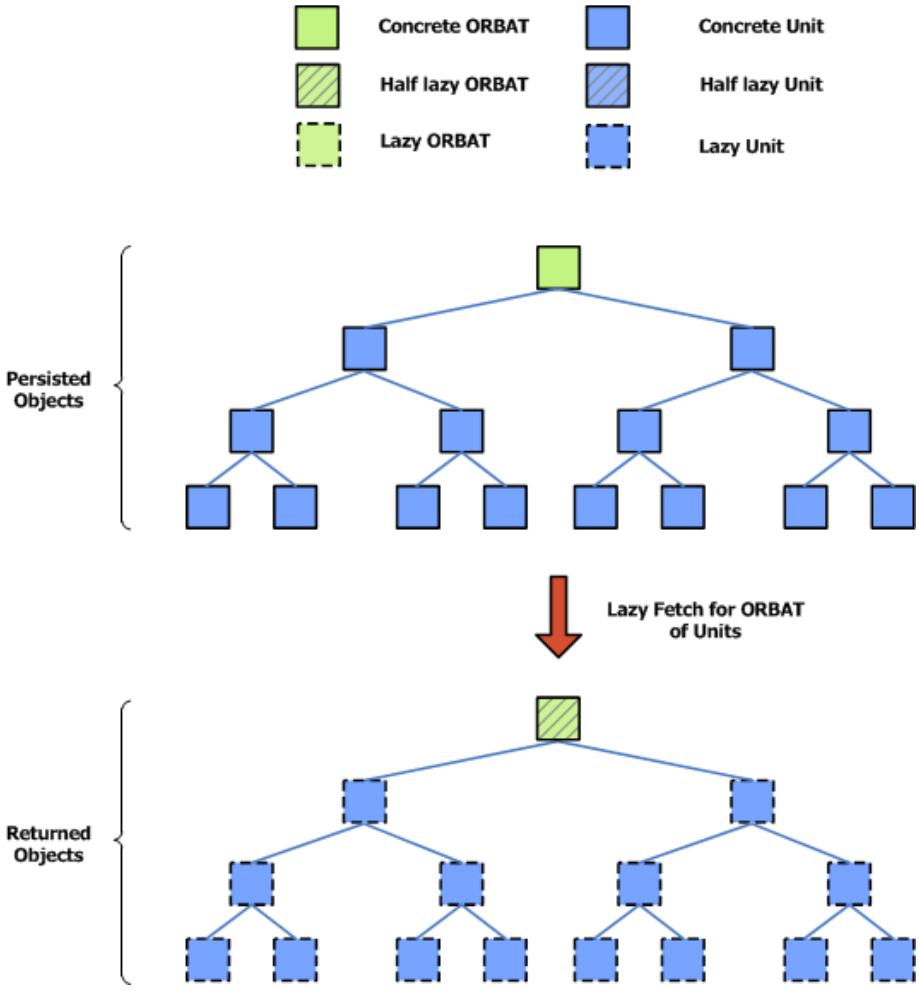

Figure 22: Illustration of the lazy loading of an ORBAT of Units

6.4.2 ORBAT of ORBATs

For want of a better term, an ORBAT of ORBATs (OO) represents an ORBAT which contains other ORBATs in a command relationship. This is useful when modelling, for example, a joint task force or coalition force structures. These embedded ORBATs may themselves contain other ORBATs or could contain just individual Units. In part this depends on how the ORBAT is being modelled for the purpose at hand. When lazily loading a structure of this type the requested ORBAT will be loaded and dependent ORBATs will be lazily loaded without a populated list of units or links. These ORBATs can be loaded by making further requests to the service. This case is illustrated in Figure 23.

Basically, the requested ORBAT will be returned as a concrete object and any dependent or child elements will be lazily loaded.

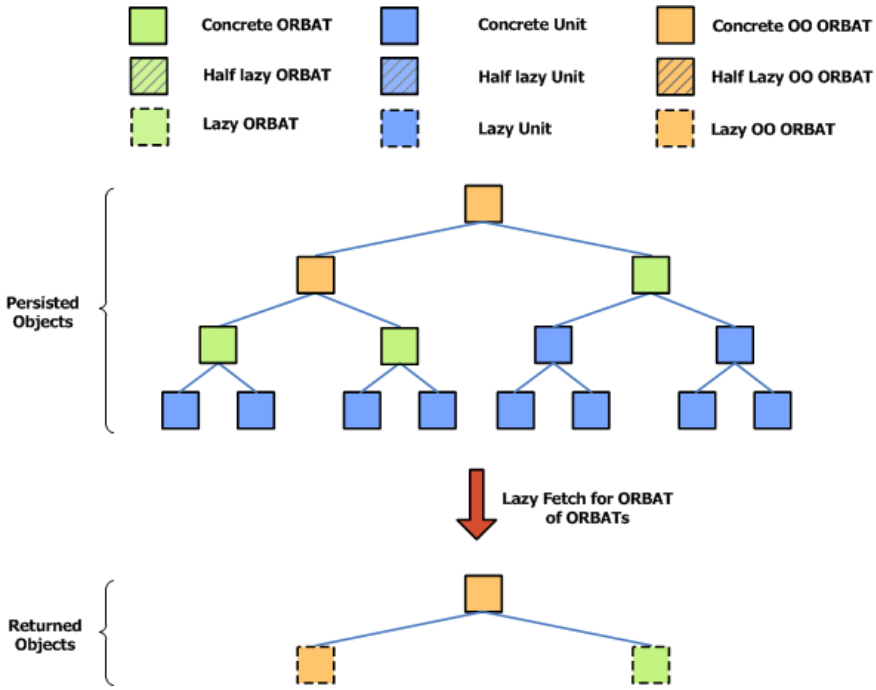

Figure 23: Illustration of the lazy loading of an ORBAT of ORBATs

7. Service Interface

The ORBAT data service has been designed with two separate Simple Object Access Protocol (SOAP)[32] interfaces, one is a general public interface for client systems to consume data from the service while the other is an administration interface used to write data to the service. This separation allows general users to gain read-only access to the service while hiding away destructive functions behind the administration interface, which can only be used by authenticated users. The permissions for the administration interface can be tightly controlled.

The authoritative definitions for these the general and admin interfaces can be accessed via HTTP from any instance of the service at:

- `http://<hostname>:<port>[/<context path>]/ORBATimplService/?wsdl` and
- `http://<hostname>:<port>[/<context path>]/ORBATadminImplService/?wsdl`, respectively.

These define the two key endpoints for the service where `ORBATPort` and `ORBATadminPort` have the implementations `ORBATimpl` and `ORBATadminImpl` written

in Java EE, respectively. The definitions for the interfaces are specified in Web Service Description Language (WSDL) [33].

The following sections outline the general and administration interfaces along with the operations that they make available.

7.1 General Interface

The ORBATDS `ORBATPort` provides read-only access to ORBAT data including search functionality and is the primary interface used by client systems. Both ORBATs and Units have a corresponding **get** operation that is able to retrieve either a single data item or a set of items. In addition, several **search** operations are provided that apply to ORBATs, Units or across both of them.

7.1.1 getORBAT/getUnit

This operation enables an ORBATDS client to retrieve ORBATs/Units from the service. A class defined in the SOAP XML schema called `GetReqType` is used to define the query to the service. ORBAT requests use a `GetORBATReqType` which extends `GetReqType` and adds the additional field `fetchDependencies` to determine the behaviour for fetching dependencies. If `fetchDependencies` is `true` then the service will fetch all child ORBAT dependencies, if set to `false` only the requested ORBAT will be returned and if it is set to `LAZY` then the ORBAT's dependencies will be lazy loaded. Lazy loading is described above in Section 6.4. Unit requests just use the base `GetReqType` which doesn't have, nor require, the `fetchDependencies` field.

A number of use cases are handled by this method as explained in Table 4, along with the field combinations required to perform the get. When performing a get the user needs to specify which temporal version they want to retrieve, as shown in the 'latest' column. More information about the temporal versioning of entities which explains the meaning of current, latest and instance can be found in Section 6.1.

Table 4: A list of request field combinations required to perform a particular operation. A dash (-) indicates that the field shouldn't be populated. The `fetchDepdendencies` field is only present for ORBAT requests. A star (*) in the `fetchDepdendencies` column indicates that the field is ignored since all requests will be lazy.

Operation	Description	Entity Types	iid	id	vid	rev	head	latest	Fetch Dependencies
Get instance	This is used to get a specific version of an entity.	Unit, ORBAT	Yes	-	-	-	-	INSTANCE	FALSE
Get instance (with dependencies)	Get a specific version and its related ORBATs.	ORBAT	Yes	-	-	-	-	INSTANCE	TRUE
Get instance (with lazy dependencies)	Get a specific version of the entity and lazily fetch its dependencies.	ORBAT	Yes	-	-	-	-	INSTANCE	LAZY

Operation	Description	Entity Types	iid	id	vid	rev	head	latest	Fetch Dependencies
Get current version	Get the current version of an entity, the head of the current version will be fetched.	Unit, ORBAT	Yes	-	-	-	-	CURRENT	FALSE
Get current version (with dependencies)	Get the current version and its related ORBATs.	ORBAT	Yes	-	-	-	-	CURRENT	TRUE
Get current version (with lazy dependencies)	Get the current version of the ORBAT and lazily fetch its dependencies.	ORBAT	Yes	-	-	-	-	CURRENT	LAZY
Get latest version	Get the latest version of an entity, the head of the latest version will be fetched.	Unit, ORBAT	Yes	-	-	-	-	LATEST	FALSE
Get latest version (with dependencies)	Get the latest version and its related ORBATs.	ORBAT	Yes	-	-	-	-	LATEST	TRUE
Get latest version (with lazy dependencies)	Get the latest version of the ORBAT and lazily fetch its dependencies.	ORBAT	Yes	-	-	-	-	LATEST	LAZY
Get specific revision	Get a specified revision of a version. Useful for traversing the revisions.	Unit, ORBAT	-	-	Yes	Yes	-	INSTANCE	*
Get head revision of a version	Get the head revision of a particular version.	Unit, ORBAT	-	-	Yes	-	TRUE	INSTANCE	*
History fetch (version)	Return all revisions of a particular version of the entity to the client.	Unit, ORBAT	-	-	Yes	-	-	INSTANCE	*
History fetch (all)	Return all revisions of all versions of an entity.	Unit, ORBAT	-	Yes	-	-	-	INSTANCE	*

An example of using the get operation in Java is seen in Figure 24:

```
GetORBATreq req = new GetORBATreq();
GetORBATReqType reqt = new GetORBATReqType();
reqt.setFetchDependencies(FetchEntityDependenciesEnum.TRUE);
reqt.setLatest(FetchEntityEnum.CURRENT);
reqt.setIid(iid);
req.getORBATid().add(reqt);
GetORBATres results = orbatdsproxy.getORBAT(req);
```

Figure 24: A snippet of Java code for making a request to get the current version of an ORBAT and its dependencies

7.1.2 searchORBAT/searchUnit/search

These operations provide an ORBATDS client with the ability to search over ORBATs and/or Units in the service. There are operations to separately search each entity type or a combined search over both entity types. It is possible to refine the search using many different criteria.

The search takes a `SearchCriteriaType` object to define the search parameters. Searching is a very important part of the ORBATDS as a powerful search helps improve the usability of the service and promotes user uptake. The criteria for the various types of search functionality are described below.

- Entity name search
- Type/Structure Type filter
- General string search across all fields
- Specific field (or field combination) search
- Current/latest search
- Association search for entities associated with a specific entity
- Orphan search
- Temporal search for entities in a specific time period.

Each of the different searches can be performed by populating the relevant fields on the `SearchCriteriaType` object and although the criteria for the different searches can be combined in different ways, only a limited number of combinations are supported. For example, an association search cannot be combined with any other search criteria. A current/latest search and a temporal search must be used mutually exclusively, but can be combined with a name, general search, structure type and/or field search criteria. If an invalid search combination is used the ORBATDS will notify the client by returning a SOAP Fault.

All of the string searches are case insensitive.

7.1.2.1 Entity Name Search

The entity name search is used to find entities in the service with a name matching a search term provided by the user. The search term is compared against the Name and Formal Name fields of the `ORBATType` or `UnitType` objects in the service (depending on which search operation is used).

Two search use cases are supported:

- **Exact Match** searches for entities which have exactly the name or formal name provided. This is the default behaviour and nothing special has to be done to do this. For example, if searching for "foo", entities with exactly the name "foo" will be returned, but entities with the names "fooaaabbb", "aaafoobbb" or "aaabbbfoo" won't be.
- **Fuzzy Match** searches for entities whose name or formal name contains the search term. To get this behaviour, the user must prefix and/or postfix wildcards to the search term. This will replicate a 'match anywhere' search, so entities with names

which contain the search term anywhere in their names will be returned. For example, if searching for "*foo*", entities with the name "foo", "fooaaabbb", "aaafoobbb" or "aaabbbfoo" will be returned. The wild card character '*' matches zero or more characters in the name and the service also accepts '?' to substitute any single character. These characters were chosen as they are likely to be the most familiar to users. To search for wild card characters themselves they can be escaped by prefixing a '\'. For example "*" can be used to find entities with the name "*" and "***" can be used to find entities with a '*' anywhere in their names.

7.1.2.2 Type/Structure Type Filter

The Type/Structure Type filter allows only entities of a particular `type` (in the case of `UnitType` objects) or `structureType` (in the case of `ORBATType` objects) to be searched over. It is possible to specify multiple filters in a single search which are combined with an OR operator. The search will return elements with the same structure type as any of those specified. The allowed values for `type` and `structureType` can be found in Section 4 by looking at the description of `UnitType.type` and `ORBATType.structureType`.

As an example, if an ORBAT search is performed and the structure type list in the search contains two elements, `OO` and `UE`, then only `ORBATType` objects with the `StructureType` of `UE` or `OO` will be returned. If a unit search sets the type to `INSTANCE`, only `UnitType` objects with the type set to `INSTANCE` will be returned.

When performing a general search, that is using the `search` operation rather than the more specific `searchUnit` and `searchORBAT` operations, it is possible to combine `StructureType` and `Type` filters in a single search request. This search will return only `UnitType` and `ORBATType` objects which have a matching `type` and `structureType`, respectively.

7.1.2.3 General Field Search

This general field search is across many different fields in a single operation to find entities which have a matching string value. More specifically, this string will be compared to values in the `name`, `formalName`, `description`, `primaryCapability`, `role`, `battleDimension`, `echelon` and `affiliation` fields provided they are not explicitly specified in the search criteria (i.e. they are null).

If this search is combined with a specific field search, the general search will be matched across the other fields. For example, if the user specified a `FieldCriteria` in the field search with `echelon` set to "company" and a `generalSearch` set to "Bob", the service will look for entities with an echelon of "company" and a name or formal name or primary capability or battle dimension or role or affiliation which matches "Bob".

This field supports wild cards as explained in Section 7.1.2.1.

7.1.2.4 Specific Field Search

The specific field search is used to find entities which have particular values in specific fields and is defined using a `FieldCriteria` object.

The fields that can be specified are:

- Capability
- Echelon
- Battle Dimension
- Affiliation
- Role
- MIL-STD-2525B Symbol Code
- Service.

These fields can be combined or omitted as desired.

The `echelon`, `battleDimension`, `affiliation` and `service` fields all have enumerated values that can be used to find entities with an exact matching value. When used in combination a quite fine grained search can be performed. The `capability` criterion takes a string which is matched against capabilities defined in MIL-STD-2525B [31] and entities which have the specific primary capability or a more specialised variant of the capability are returned. This is because MIL-STD-2525B defines a hierarchy of capabilities. The further down the hierarchy the more specialised and fine grained the capability, while higher nodes are more general.

For example, let's say there are three Units. The first two Units have the capabilities of "CARGO AIRLIFT (MEDIUM)" and "CARGO AIRLIFT (HEAVY)" respectively. These capabilities are leaf nodes in the capability tree. They both have the same parent, "CARGO AIRLIFT (TRANSPORT)", which itself has the parent "FIXED WING". The third unit has the capability "FIGHTER" which is also a child of "FIXED WING".

If a capability search is performed with the string "CARGO AIRLIFT (MEDIUM)" only the first unit will be found. If a search were performed with either "CARGO AIRLIFT (TRANSPORT)" or "CARGO AIRLIFT" then the first two units will be returned. If a capability search was performed with "FIXED WING" then all three units would be returned as they have "FIXED WING" as an ancestor of their capabilities.

Something to note is that capability names in MIL-STD-2525B are not unique. For example, there are aircraft with a cargo capability, as well as ships with a cargo capability—both of these capabilities are different, but have the same name. Consequently, when performing capability searches entities returned may not match the desired capability exactly.

The `capability` criterion is also used to find entities which list a matching capability as one of their secondary capabilities. In addition this criterion can also be combined with some flags to control its behaviour:

- If `primaryOnly` is true, then only the primary capability field will be used in the criterion—secondary capabilities won't be included in the search.

- If `exactCapabilityMatch` is true, then only entities whose primary capability string matches the search term will be returned. The MIL-STD-2525B capability hierarchy will not be used.

These two flags make it possible to create a lazy loaded ORBAT explorer which groups entities based on their primary capability. It also makes it easier to find custom capabilities which are not a part of the MIL-STD-2525B as it reduces the number of false positive matches. Many of the capability names preferred by the ADF can not be found in MIL-STD-2525B making custom, non-standard capability descriptors necessary.

The MIL-STD-2525B Symbol Code criterion allows searching for entities which have a particular symbol code. Searching with the symbol code '`SHGPEWA—HAU`' will return all entities which have that exact symbol code. The field also supports wild cards so specifying '`SHGPEWA---???`' will find entities which have symbol codes that match the first 7 characters of the search term. A '?' matches any single character. In MIL-STD-2525B each of the characters in the symbol code is used to encode something about the entity, so using this search functionality it is possible to perform a wide variety of searches. For example, you can find every entity of a specific echelon by specifying only the echelon in the symbol code and having a '?' in place of all other characters.

If looking for a specific capability, using the symbol code search to find that capability will be more accurate than using the free text capability search.

7.1.2.5 Current/Latest Search

The search can be configured to search for either the current or latest versions of entities by specifying `GeneralCriteria` in the search request. Using Figure 25 as an example, to search for the latest version of an entity set `latest` to true in the criteria. In this case Revision 2 of Version 2 will be returned to the client as this is the head revision of the latest version of that entity. To search for the current version of an entity the `latest` variable must be false. This will return results whose version intersects the current time that the query was made. In this example revision 3 of version 1 will be returned.

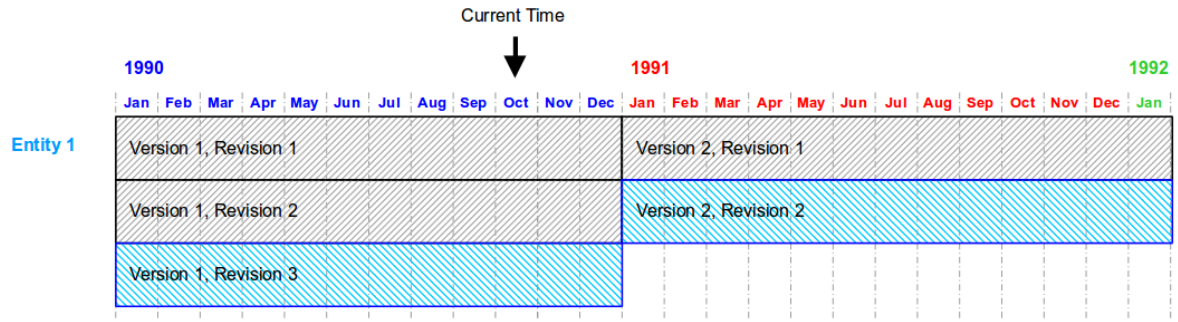

Figure 25: A temporal diagram of an entity showing two versions with multiple revisions. The head revisions are shaded in blue.

7.1.2.6 Association Search

The association search is used to find all ORBATs that are associated with a particular ORBAT or unit in the service. An entity is associated with another if it has been used in a larger ORBAT that contains it. The client specifies the entity's unique instance identifier, `iid`, for which they want to find all associated entities.

For example, consider the user has a unit, Unit 1, and they wish to find all ORBATs in the service which contain that unit. To do this the client performs a unit association search for all ORBATs which contain that unit, an example of which can be seen in Figure 26.

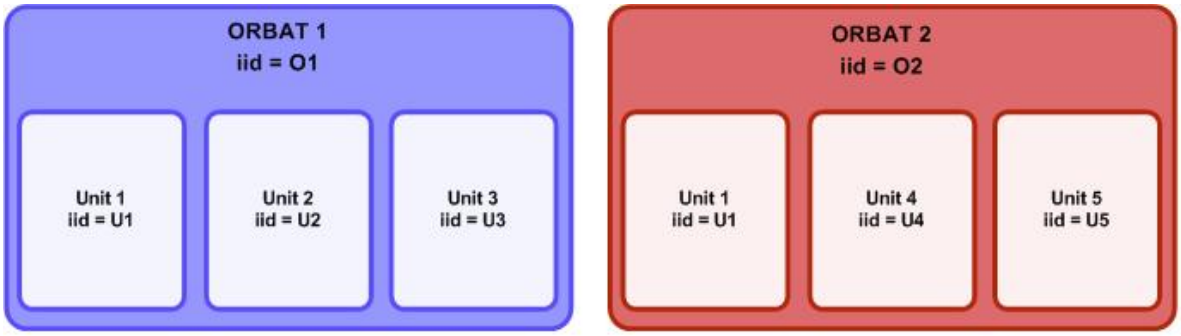

Figure 26: When performing a unit association search for Unit 1 the result will contain ORBAT 1 and ORBAT 2

A similar search can be performed to find ORBATs which are associated with another particular ORBAT.

7.1.2.7 Orphan Search

The orphan search option is used to restrict a search to only return orphan, or unallocated, units. These are entities which have not been used in any ORBAT. This search option is useful when creating new data because among the many new units that have been assigned to an ORBAT, some may still remain unassigned.

The orphan search is specified by setting the `orphansOnly` boolean in `SearchCriteriaType` and can also be combined with other criteria.

7.1.2.8 Temporal Search

The temporal search can be used for more fine-grained searching of entities across time. It allows clients to find versions of entities which exist either at a particular point in time or within a period of time. To search for entities which exist at a particular point in time the client simply provides this time in the temporal criteria of the search request. Any entities which intersect this point in time will be returned to the client. The temporal criteria can be combined with other searches and it is recommended to do so as the number of results can often be quite large.

An example of a point in time search is shown in Figure 27. If the client were to specify the point in time as being 1st of April, 1991 and combined it with the name criteria "Entity 3"

then version 2 of that entity would be returned. If the name criteria was not specified then version 1 of Entity 1, version 2 of Entity 2, version 2 of Entity 3 and version 1 of Entity 4 would all be returned.

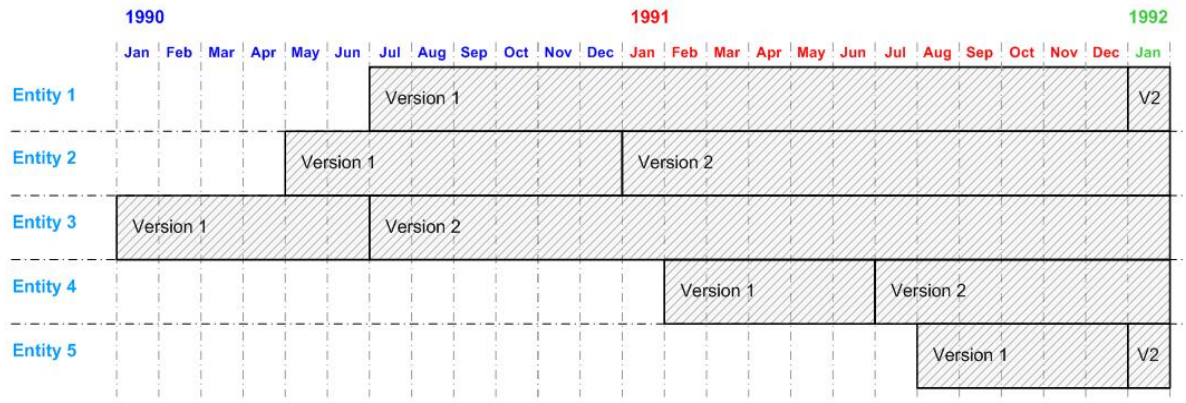

Figure 27: A temporal diagram showing different versions of a number of entities over time

The temporal search also supports period searching. For this search the client specifies the start time and finish time of the period they are interested in. A modifier can be used to specify the behaviour of the period search to either return only those entities which start within the period or to return all entities which exist for any part of the specified period. For Figure 27, if the user specifies the period of 1st June 1990 to 15th Jan 1991 and selects to return only entities which start in this period then version 1 of Entity 1, version 2 of Entity 2 and version 2 of Entity 3 will be returned. If the user did not restrict the search to only those that start in the period then version 1 of Entity 1, version 1 & 2 of Entity 2 and version 1 & 2 of Entity 3 will be returned.

7.1.3 summariseUnits/summariseORBATs

Given a list of units or ORBATs, the service will summarise the entities by aggregating all of the Aide Memoire (AM) objects that they contain such as equipment, supplies and personnel. If the client is only interested in aggregating a subset of the AM objects they are able to provide a list of AM identifiers, otherwise if no identifiers are selected then all items are aggregated. The operation will expand any cardinality of any subordinate units prior to calculating the summary.

Units which are present more than once in the request will only have their AM objects counted once in the aggregation. For example, Figure 28 shows three ORBATs that are to be passed to the summariseORBATs operation. In this case the AM objects from all units in these ORBATs will be aggregated, but the AM objects of unit D will only be counted once.

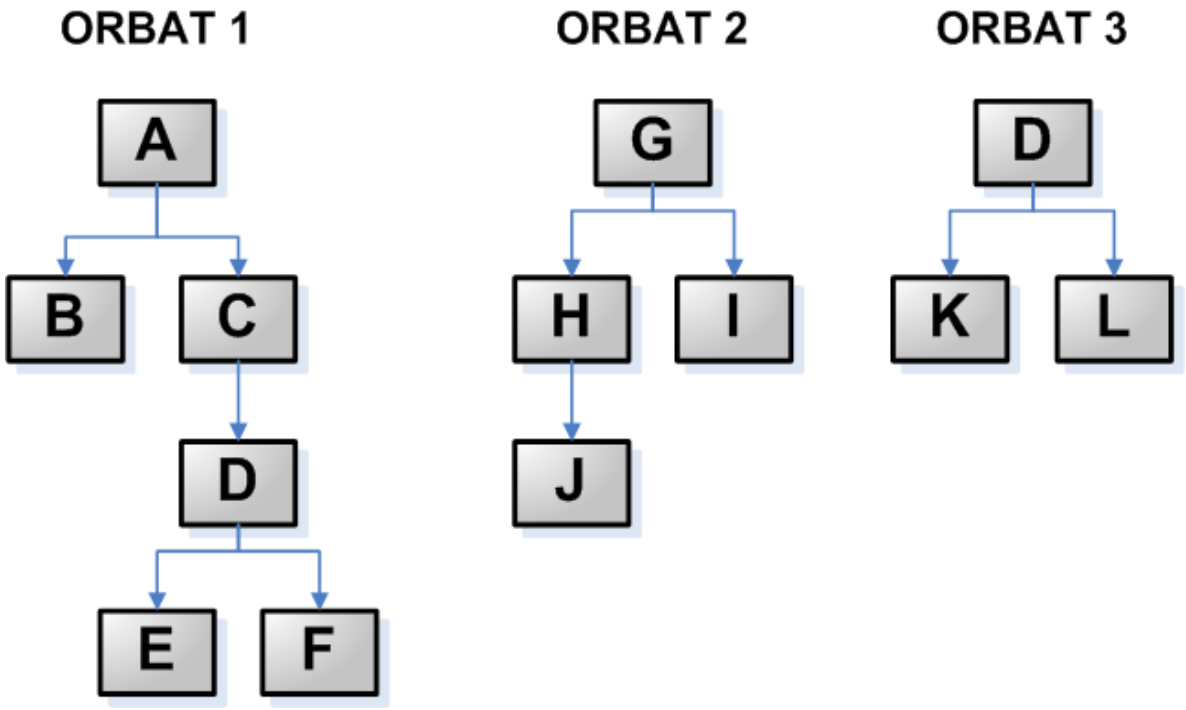

Figure 28: Three ORBATs to be summarised. Unit D is common between two ORBATs, but its AM objects will be counted just once

If any of the ORBATs within the request contain cardinality, such as ORBAT 3 in Figure 29, then it is assumed that all of the units in the request are gerneric capability bricks and can therefore be counted multiple times. In Figure 28 there was no cardinality so each of the units was only counted once. In this case, when summarising the Aide Memoire objects unit D will be counted twice since it appears in ORBAT 1 and ORBAT 3 and cardinality is present in the request. AM objects from units K and L will be counted 3 times each and the other units will be counted once.

The ORBATDS returns to the client the aggregated list of Aide Memoire items.

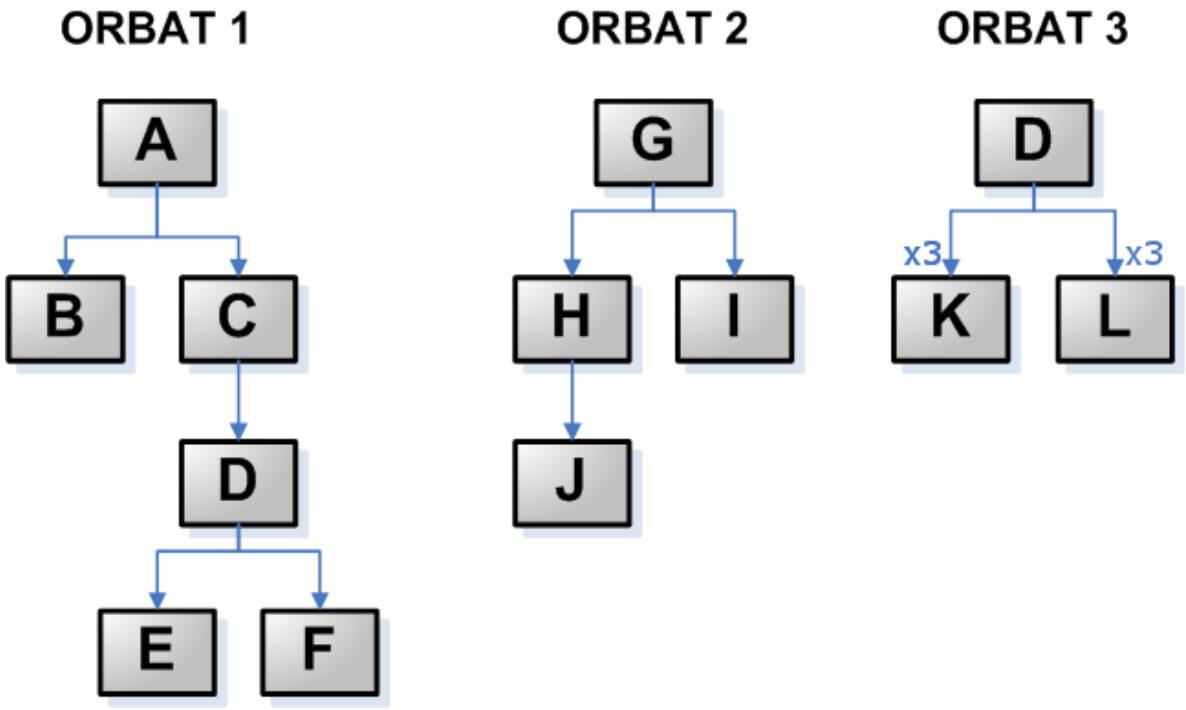

Figure 29: Three generic capability bricks to be summarised. ORBAT 3 contains cardinality on its links meaning that units are counted multiple times.

7.1.4 summariseUnitsExpanded

This summarising operation is used when a summary of a selection of units plus any of their subordinate units is required. In other words, it computes the total number for each type of equipment, supplies and personnel that are contained in entire command trees extending from selected units. A flag is used to determine whether duplicate units existing across multiple ORBATs are allowed to be aggregated more than once as the command trees are traversed.

This method has a few potentially confusing cases and so the following figures describe the desired behaviour in some of those cases. A simple case is shown in Figure 30 in which units A and D in ORBAT 1 are selected to be summarised, but they happen to be in the same ORBAT. In this case each unit including its subordinates are aggregated once.

In Figure 31 Unit D in ORBAT 1 has been selected for summarising. Unit D also happens to be present in ORBAT 3. As a result of this request units D, E, F, K and L will be aggregated. If the duplicates flag is true, then unit D will be counted twice.

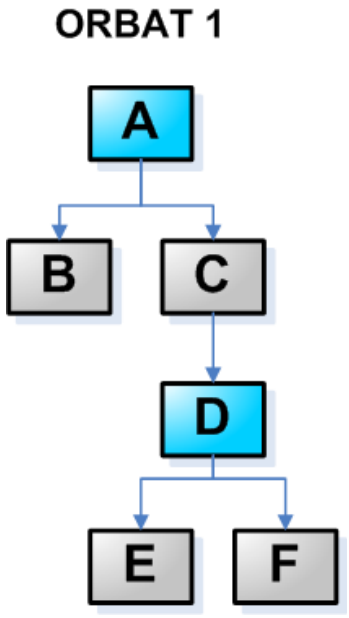

Figure 30: Two units have been selected to be summarised that happen to be in the same ORBAT. Units A and D are counted once.

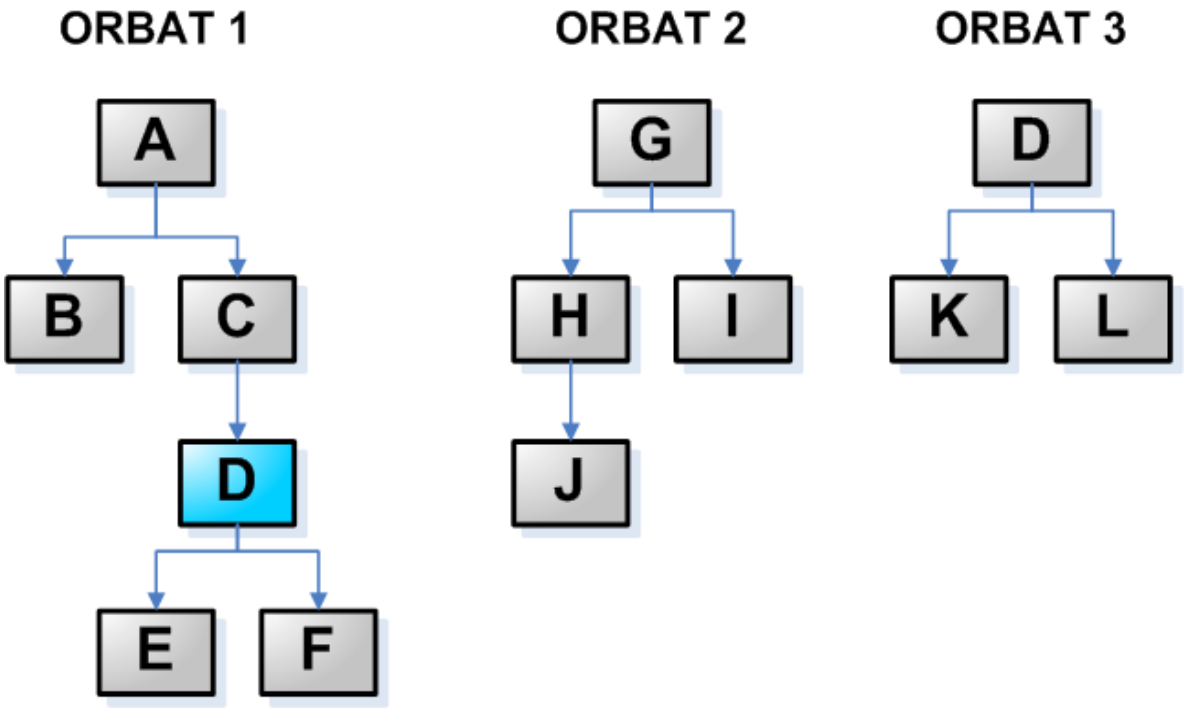

Figure 31: A request for aggregation containing three ORBATs with Unit D from ORBAT 1 selected

Figure 32 shows a request identical to that shown in Figure 31 with the exception that ORBAT 1 is a generic capability brick with cardinality. In this case, Unit D would be aggregated 4 times regardless of whether the duplicates flag is set or not. Units E & F

would be aggregated 3 times (as they appear 3 times each once the cardinality has been expanded) and units K and L would be aggregated once.

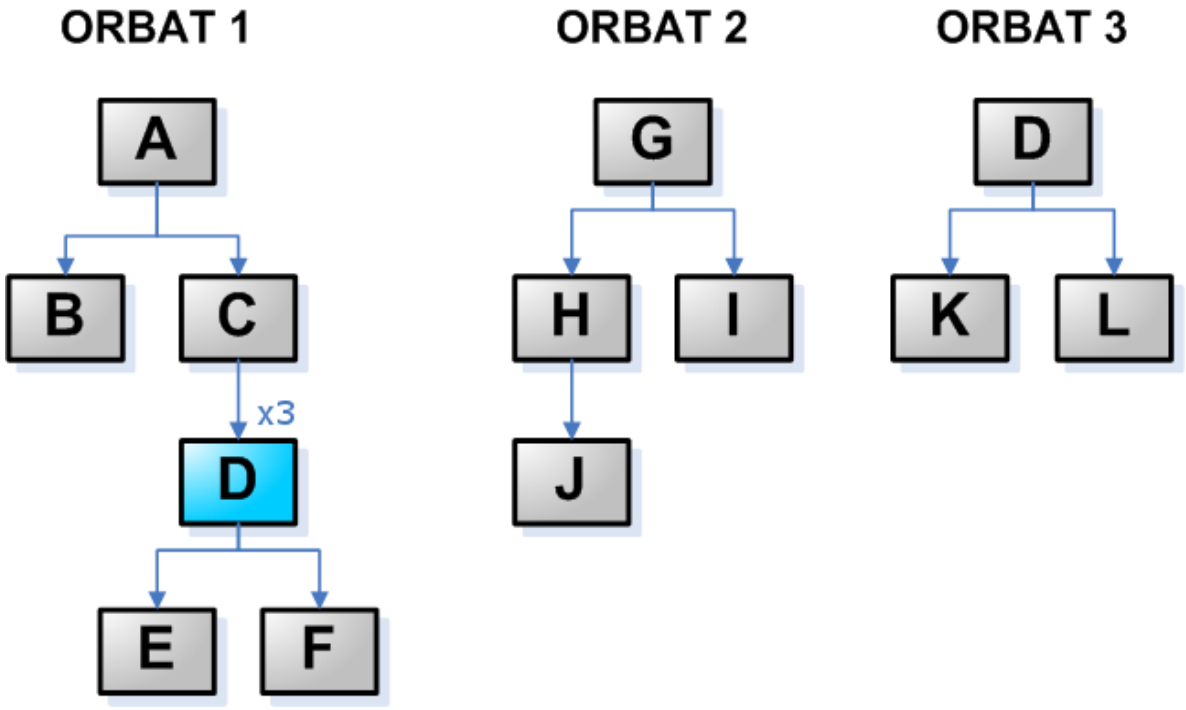

Figure 32: A request for aggregation containing three ORBATs with Unit D from ORBAT 1 selected. ORBAT 1 contains cardinality.

Finally, in Figure 33 units B, C from ORBAT 1 and unit G from ORBAT 2 have been selected for summarising. ORBAT 1 is again a generic capability brick specifying cardinality on one of its links. In this case units C, D, E and F will each be aggregated 3 times regardless of the duplicates flag and units B, G, H, I and J will each get included in the summary just once.

The ORBATDS returns to the client the aggregated list of Aide Memoire items including equipment, supply items and personnel.

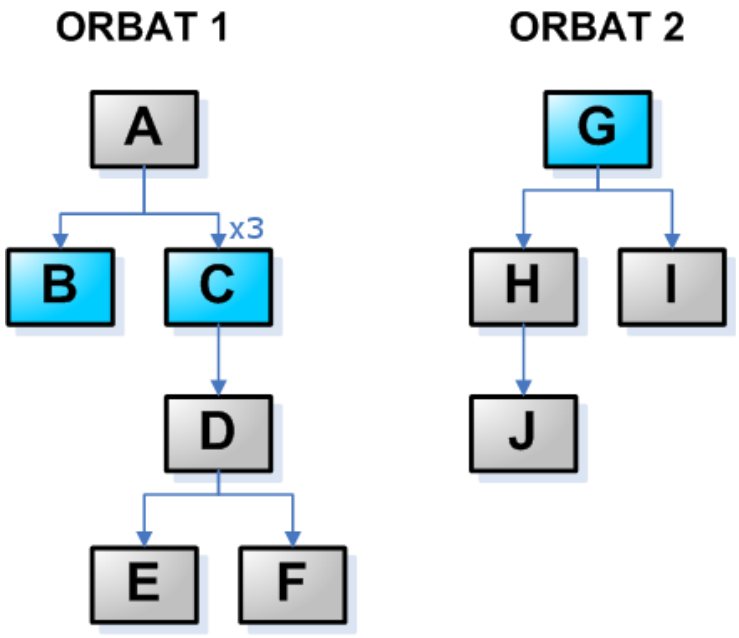

Figure 33: A request for aggregation containing two ORBATs with three units selected. ORBAT 1 contains cardinality.

7.1.5 getUnitSummary

This operation can be used for two things. It can be used to pull a unit and its subordinates out of an ORBAT and return the selected sub-hierarchy, which is useful when the client doesn't require an entire ORBAT. It can also be used to return a summary of a unit and its children, aggregating all of the Aide Memoire items into a single unit for the client.

This operation takes three parameters. The first is the unique identifier, `iid`, of the ORBAT from which the hierarchy is to be taken, the second is the `iid` of the root unit that the client is interested in and the third is a boolean flag used to determine whether to return the sub-hierarchy or to return a summary.

7.1.6 get2525Symbol

This operation is used to leverage the ORBATDS support for MIL-STD-2525B. The client can provide values for the fields that determine the symbology such as capability, echelon, battle dimension etc. The service will then return to the client the matching symbol as well as its 15 character symbol code.

7.1.7 listCapabilities/listPrimaryCapabilities

These operations return to the client a list of all the unique primary capabilities which have been assigned to the head revision of the specified version (i.e. current or latest) of all entities in the service. The entities used to build the list can be filtered by military service (e.g. Army), unit type (e.g. brick or instance), ORBAT type (e.g. capability brick or unit entitlement) and whether to only include orphan/unallocated units. The capabilities are

returned either as MIL-STD-2525B with Change 2 symbol codes or as capability names depending on which operation is used.

These methods can be used to build up an explorer hierarchy to enable the client to browse the data via capabilities.

7.2 Administration Interface

The ORBATDS `ORBATadminPort` service provides authenticated users with read/write access to ORBAT data where writes are managed through a three stage data management process. The service provides **put** and **deprecate** operations. The put operations allow privileged users to create new data entries or to update existing ones. The deprecate operations mark entities as being no longer required and is similar in function to a deletion process except that the data is retained for audit and maintenance purposes. Deprecated data is not visible in search results by default from the `ORBATPort` service.

To gain access to the administrative operations the client must first authenticate with the service to gain access. Access is controlled through the web application server such as Tomcat or WebSphere. In the server, users or groups of users are assigned different roles and jurisdictions in order to edit data and perform administrative duties. More information about this can be found in Section 9.

7.2.1 putORBAT/putUnit

These operations are used to store user defined ORBATs/units in the service. The user first builds up the ORBAT and unit data locally and then uses the put operations to put it into the service. The data must pass validation rules before the service allows it to be persisted, as discussed in section 10.4.

These operations are also used to make modifications to existing data. Once data has been retrieved from the service the user can make changes to the data and push it back to the service to create a new version or revision of the entity. When making modifications to an entity it must be a draft in the *EDITED* state resulting from a fetch using the getDraft operations. If the start date is changed then a new version of the entity is created. If there are no changes to the start date then the entity is stored as a new revision of the existing version.

To make a change to the start date without creating a new version, for example to correct it, the `modifyDateOnly` flag can be set on the request. This can only be used to change the start date on the latest version of an entity.

7.2.2 depORBAT/depUnit

These operations are used to deprecate entities that are no longer used or otherwise useful. The system does not provide the ability to completely delete an entity to enable other

entities which reference it to continue doing so. This ensures that all existing entities continue to function while discouraging future use of the deprecated entity.

7.2.3 getDraftORBAT/getDraftUnit

These operations are used to get a draft version of an existing entity, as mentioned above. These operations will return a draft of the latest revision of the specified version for the client to edit.

7.2.4 updateState

This operation is used as part of the data management process to transition entities through the different states of that process. This is used to transfer drafts between EDITED, AWAITING_VERIFICATION, VERIFIED and APPROVED as explained in Section 9.

7.2.5 searchORBAT/searchUnits/search

The administrative search offers the same functionality as the general search, but also enables the client to specify the state of the entity they are searching for rather than just restricting the search to APPROVED entities. This can also be used to search for deprecated entities.

7.2.6 getAuthorisedRoles

This operation is used to find the list of roles that the authenticated client has been assigned. This can then be used by client applications to determine which functionality to expose to the user.

7.2.7 getUserJurisdiction

This operation is used to find the jurisdiction of the authenticated user. The jurisdiction affects what entities the client has permission to modify and thus can be used by client applications to modify functionality accordingly.

7.2.8 getRepositoryID

This operation is used to determine the repository ID of the ORBATDS which the client is authenticated with.

7.2.9 listCapabilities/listPrimaryCapabilities

These operations offer the same functionality as the operations on the general interface, except that it also allows the client to specify the state of entities as an additional filter.

7.3 REST Interface

In addition to the SAOP Web Service interface described above, there is also a very limited set of functionality offered via simple REST interfaces. This is mostly used by the administrative reports, but can also be used in other applications. Table 5 lists the different REST resources that are provided by the ORBATDS.

Table 5: Description of the different REST resources that are available

Path	Function
`/<context path>/orbat/<iid>`	Get an ORBAT specified by IID. This interface returns a single ORBAT in XML format. When requested in a browser an XML Style Sheet is used to transform the XML into HTML for viewing.
`/<context path>/unit/<iid>`	Get a unit specified by IID. This interface returns a single unit in XML format. When requested in a browser an XML Style Sheet is used to transform the XML into HTML for viewing.
`/<context path>/symbol/<symbol_code>`	Generate a symbol from the 15 character MIL-STD-2525B symbol code. It also takes width and height query parameters to specify the size of the symbol you want returned. The returned symbol is in the PNG format. This is useful for web pages which wish to use military symbology.

7.4 ORBAT Administration Client

The ORBATDS shipped with an associated ORBAT Administration Client (ORBATAC) to administer the data in the service. This web-based client uses the ORBATDS Administration interface to create and modify entities and implements the data management workflow described in Section 9. It is implemented as a Java EE Web Application and uses JavaScript to implement the interface behaviour. Figures 34 and 35 show the ORBATAC. The left most pane is the ORBAT explorer. This is used to search and browse through the entities stored in the ORBATDS. The right-most pane is the Aide Memoire Explorer. This allows users to search and browse for containers, equipment and persons stored in the AMDS that can be used when constructing units. The centre pane is the main editor. It enables all aspects of the entity to be edited. Along the top are buttons used to move the entity though the data management workflow, and also to create new entities.

A number of shortcomings have been identified with this administration client that are described in Section 11.6.

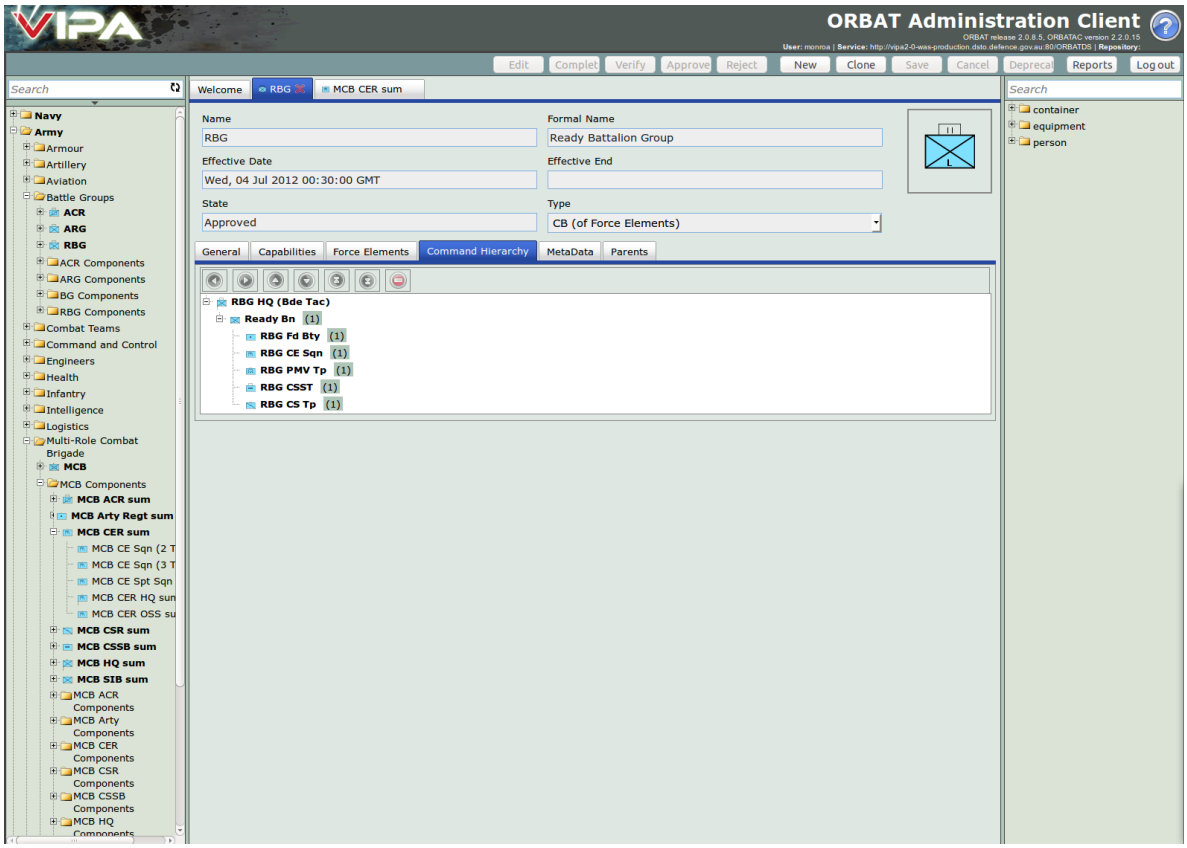

Figure 34: A screenshot of the ORBATDS Administration Client (ORBATAC) showing the command hierarchy editor which is used to configure the command links in an ORBAT

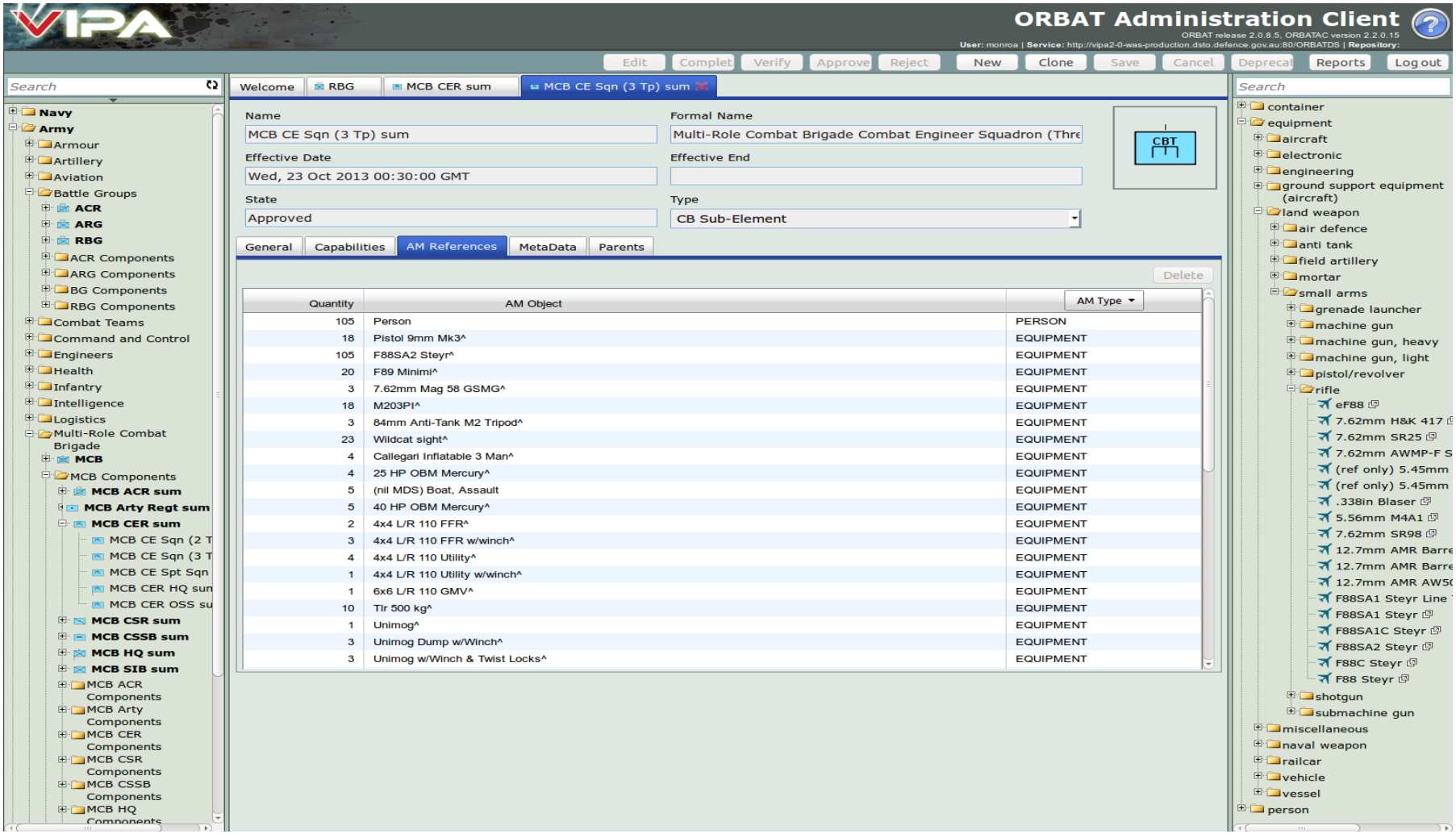

Figure 35: A screenshot of the ORBATAC showing the Aide Memoire References editor. This is used to configure the containers, equipment and persons a unit holds. Aide Memoire items can be dragged from the explorer on the right and dropped into the unit's holdings.

8. Design

8.1 Design Patterns

The ORBATDS makes use of several software design patterns as described by the "Gang of Four" [21]. The use of design patterns enhances the development process by providing tested, proven coding paradigms with improved code readability and speeds up the development process. The main design patterns used in the ORBATDS are:

- Composite: Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
- Factory: Deals with the problem of creating objects without specifying the exact class of object that will be created. Achieved by defining an interface for creating an object, but letting the subclasses decide which class to instantiate.
- Singleton: A design template that allows only one object of a class to be created, and provide a global point of access to it. Used where ownership of the single instance cannot be reasonably assigned, lazy initialization is desirable, and global access is not otherwise provided for.
- Iterator: Provide a way to sequentially access the elements of an aggregate object without exposing its underlying representation.

8.2 Technology

This section outlines some of the technologies used by the ORBATDS.

The ORBATDS is a Web Service application for managing of Order of Battle (ORBAT) information that is required by planning and logistics tools such as ViPA. The service is implemented as a Java Enterprise Edition 5 (Java EE 5) [1] Web Service using JAX-WS technology that implements the JSR-224 Standard [2]. Java EE 5 was chosen specifically because it was supported by our targeted deployment server, IBM Websphere Application Server Version 7 [3]. Java EE application servers provide a framework within which security, data access and inter-application communication channels can be configured. The specific JAX-WS implementation used by ORBATDS is that found at the official JAX-WS website [4] and is also included in the official Java runtime and development environments.

Client applications can access the service using Simple Object Access Protocol (SOAP) interfaces and the REpresentational State Transfer (REST) interface for some basic functions. The SOAP interfaces described in Section 7.1 and 7.2 are defined using Web Service Definition Language (WSDL) version 1.1 and will simplify interoperability with other systems which also implement that SOAP standard. The REST interface merely exposes a subset of functionality in the primary SOAP interface that are beneficial for users, as described in Section 7.3. For example, it provides a light weight means to access

an entity via a HTTP request without the overhead of SOAP message compliance. The REST interface only returns data in XML format and uses an Extensible Stylesheet Language Transformation (XSLT) to transform that data into HTML for display in a web browser.

The ORBATDS was developed using a 'WSDL first' approach, meaning that the WSDL describing the service interface as well as the corresponding XML schema defined data model were created before developing the software. The 'WSDL first' approach postulates that the public data model, that is the model exposed though the public interfaces of the ORBATDS, is the most important. Therefore, these interfaces should be designed first and free of implementation considerations, rather than generating them from other models or straight from the application code, as is often the case with other development approaches.

Consequently, once the interface was designed, it was then possible to begin developing the client and service implementations in parallel. The implementation of the data model in the programming language (Java) was generated automatically from the WSDL service definition and XML Schema using JAX-WS with the Java Architecture for XML Binding (JAXB) [6] for XML to Java binding. The XML schema which defines the ORBAT data model can be found in Appendix A.

The JAX-WS framework consumes the WSDL ORBATDS definition (including the referenced schemas) to produce the interfaces required. Through this process, the JAXB framework creates several Java class libraries that completely represent the ORBAT data model. The complex types defined in the ORBAT.xsd XML schema file (Appendix A) therefore have corresponding Java classes. The process by which data model artefacts are generated is described by Ort (2003) [6]. Once the XML schema and WSDL binding process is complete, the resultant Java libraries are used as the basis for the ORBATDS implementation and are also used by a number of clients and utilities.

Data persistence is implemented using the Hibernate Object/Relational Mapping (ORM) [9] framework which implements the JSR-220 [10] Java Persistence API (JPA) specification. This allows the data model Java objects generated from XML schemas to be mapped into a relational database structure. The mapping allows data model object instances to be saved and retrieved from a database supported by the Hibernate ORM framework. The ORBATDS has been designed to use the Oracle 10g database, however other databases are automatically supported.

In addition, data model validation is performed using the JSR 303 [7] Hibernate Validator reference implementation [8]. The validation framework defines a set of validation rules (constraints) in a single place which can be applied across multiple system layers from the user interface through to persistent data stores. This minimises duplication of human effort and maximises consistency. The validation framework is used to prevent erroneous data from entering the system and for providing feedback to users on the data in the system.

IBM WebSphere Application Server and Oracle 10g database software is used on national fixed networks. An alternative lightweight installation using the Jetty Servlet Container

[36] and HyperSQL DataBase (HSQLDB) [37] is also supported for use on deployed laptops and networks. This enables the service to run self-contained on a single machine, which is not possible using the WebSphere and Oracle options due to their large hardware requirements.

9. Data Management Framework

High quality data is absolutely essential for users to trust in the accuracy of calculations generated by applications such as the ViPA workbench [34]. Therefore the objective of the data management framework is to ensure that the collection, input, verification, validation and management of ORBATDS data is adequate to support operational planning. The ORBATDS has a multi-stage data management process which provides release control mechanisms for ORBAT data on a per entity basis. It also guarantees that only suitably authorised users can participate in the process to minimise the chances of unsuitable data being made available to the broad Defence user base. The ORBAT data management process is complementary to the ORBAT entity versioning scheme outlined above in Section 5.

The data management framework recognises various user roles that determine how a user interacts with the data and what they are allowed to do to it. The four roles defined are: EDITOR, VERIFIER, APPROVER and REPORTER. These roles grant rights to perform different stages of the data management process as described below. Entities which have been rejected in any of the steps need to be revisited by the EDITOR.

It is important to note that the framework has been designed such that a draft entity cannot be consumed by general users, it is only when the data has been approved that it is published and then made accessible to users through the general service interface.

Figure 36 shows an overview of the ORBATDS data management process. The black circles at the top indicate the state of the entity as it transitions through the process along the valid transition paths. The blue boxes indicate the different stages of the process where a box with a dashed border indicates that only a super administrator can perform that stage. The green boxes show the software involved in each stage along with the actors which perform them.

9.1 Data Management Stages

There are a number of stages in the data management framework; this section outlines each of them.

9.1.1 Manage

This stage is actually outside of the ORBATDS and so is beyond the scope of the data management process. However, it is included for the sake of completeness since it is

necessary to support that workflow. This stage requires the ADF owners of various data sets (e.g. data about vessels would be owned by the Navy) to determine what data sets they should own and who needs what permissions to manage those data sets in the ORBATDS. External systems such as Active Directory would be used to manage user credentials and so this is where ORBATDS user roles would also be managed. See Section 10.1.

9.1.2 Capture

This stage finds the relevant data and retrieves it if possible. Experience has shown this is not a trivial matter as such data can be commercially sensitive, operationally sensitive (thus classified) and even organisationally sensitive. Metadata about the author of the data, where it was sourced from, the modification date of the data, etc. also need to be collected and entered along with each entity into the ORBATDS. There is also a need for reporting on the progress of data entry, particularly when contractors are being used for that purpose. This stage requires a user with editing rights given by the EDITOR group membership (see Section 10.1).

This stage begins with the creation of a draft version of either a new or existing entity. This is done when an editor calls the getDraft operation on the ORBAT administrator interface. The draft entity's state variable is set to 'EDITED'. There is no limit on the number of changes that a draft can undergo and each change is recorded as a metadata entry since the draft is not published, therefore there is no need for a new revision. However, once the editor is satisfied that the data meets the quality criteria as defined in the ViPA Data Services Management Framework (VDSMF) [35] they can complete editing and mark the entity as ready for verification. This changes the state of the entity to 'AWAITING_VERIFICATION'.

9.1.3 Verify

This stage verifies the data entered or modified during the Capture stage to check that it was obtained from the appropriate sources, that it is correct, that it is still current (i.e. not out of date) and that it has the necessary metadata. The verification stage of the process requires a user with verification rights given by VERIFIER group membership. The data to be verified is retrieved from the service using the getDraft administrator operation. At this stage the entity can be verified as correct or rejected for further editing, which is performed by invoking the updateState operation on the administration interface with either 'VERIFIED' or 'REJECTED' state values, respectively.

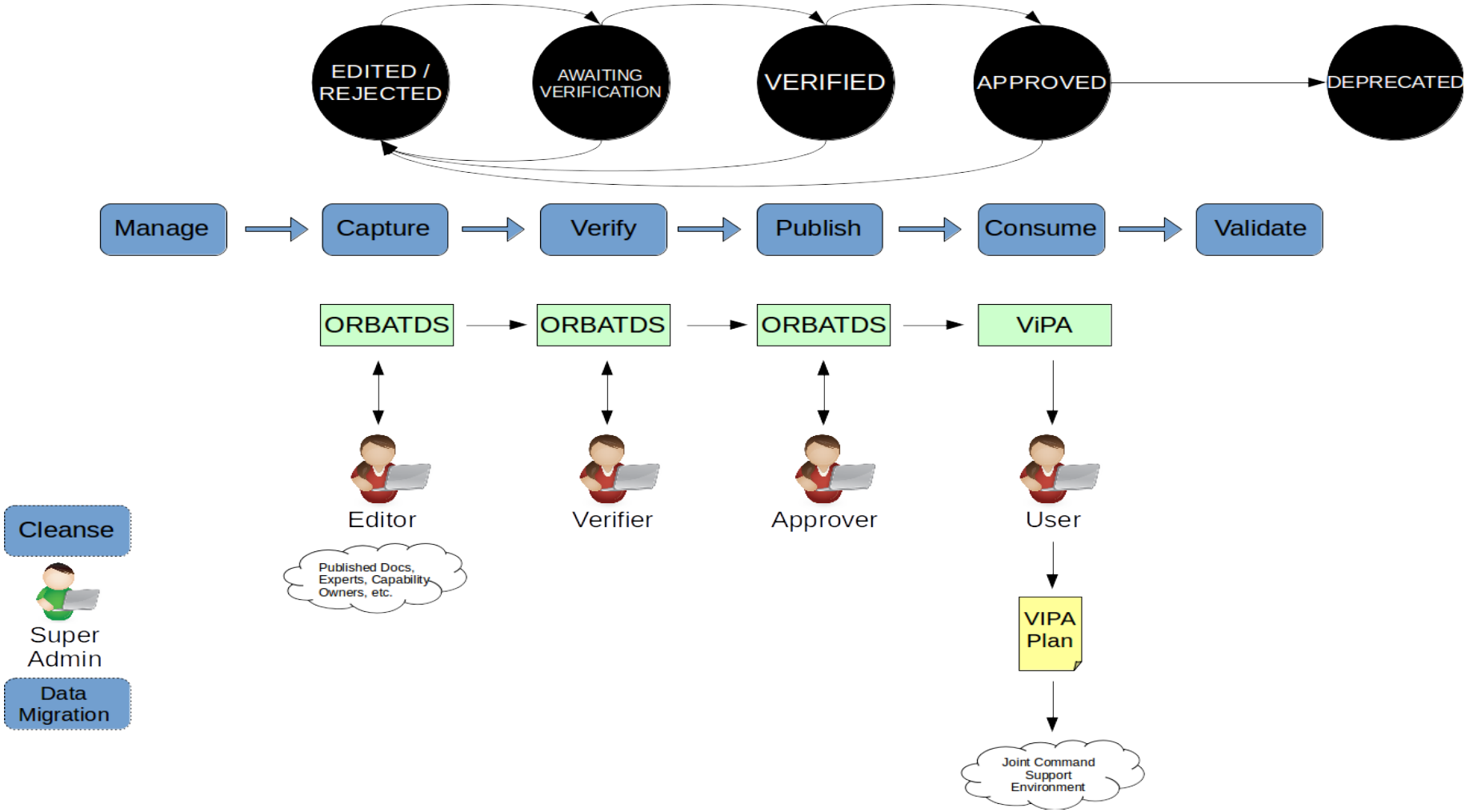

Figure 36: An overview of the ORBATDS data management process

9.1.4 Publish

The data is retrieved from the service using the getDraft operation. This user performs a final check of the entity and inokes the updateState operation on the administration interface with either 'APPROVED' or 'REJECTED' state values. On approval, the entity will be published and a new revision will be created. This new revision will now be visible to all users of the ORBATDS.

9.1.5 Consume

In this stage various applications such as ViPA use the General Interface to obtain the ORBAT data they require.

9.1.6 Validate

Data requirements are complex and continually changing, particularly during the conduct of operations. Therefore, it is not possible when performing Verify and Publish to ensure that the ORBAT data will continue to be fit for all purposes. User feedback is essential to identify where and when the data is no longer suitable for use and should trigger the Capture stage to conduct further data collection, possibly from the field.

9.1.7 Cleanse

In the event of a security spill, there is a requirement to be able to cleanse the ORBATDS data to remove and potentially replace the offending entries. This action should be carried out by a Super Administrator who has direct read/write access to the database. Cleansing will necessarily circumvent the ORBATDS audit records.

9.1.8 Data Migration

There is a need to migrate data between multiple ORBATDS instances deployed on different classification networks. The general rule is, manage data on a network that is classified (ideally) at the same level as that data or at the nearest available classification above that of the data. Then migrate that data upwards to more classified networks as required.

9.1.9 Deprecate

A user with approval rights given by APPROVER group membership is able to deprecate entities that should no longer be used. As previously mentioned, deprecation does not delete the entity, rather it hides it from the general user unless the user has an existing reference to it, for example, as part of an existing ORBAT. This ensures that deprecation can be carried out safely without impact on other current uses of the data, thus only preventing future use of the data.

9.2 Entity States

Four states have been described so far to identify where entities are in the data management process. However, an additional two states are needed to provide fine grained control over the data management process. First, it is important to make a distinction about whether an entity is being edited or the editing process has been completed. In the latter case the state of "awaiting verification" means that the editor(s) have completed updating the information of that entity. Second, it is useful to differentiate between entities that are being edited and those where the editing process has previously been completed, but the data was later rejected and so requires fixing.

Entities are transitioned through the sates described below, states cannot be skipped:

- EDITED: The entity is a draft and is not visible to general users.
- AWAITING_VERIFICATION: The entity is ready for a verifier to check for correctness.
- VERIFIED: The entity was verified and is ready for an approver to decide if it is fit for purpose.
- APPROVED: The entity was approved for publication to make it available to all users.
- REJECTED: The entity was rejected during the verification or approval stages and so needs to be revisited by an editor. This state has the same valid transitions as the EDITED state.
- DEPRECATED: The entity should no longer be used. These entities are still accessible via other existing entity references in order to maintain data integrity.

Entities which are in the EDITED, AWAITING_VERIFICATION, VERIFIED or REJECTED states are considered to be drafts and so are not exposed through the ORBATDS General Interface. Each version of an entity can only have at most one draft at any time. Drafts exist outside of the normal revision history of an entity and internally are represented with a revision value of -1. When accessing a draft of a particular entity, the version of that entity must be specified. If a draft for that version already exists then it will be returned, otherwise a new draft based on the head of that version will be created. Figure 37 illustrates an entity which has a single version with three revisions in the ORBATDS. It also has two drafts indicated by the dashed outlines. The first draft is for the existing version which, if published, will become the fourth revision. The second draft is for a new version of the entity.

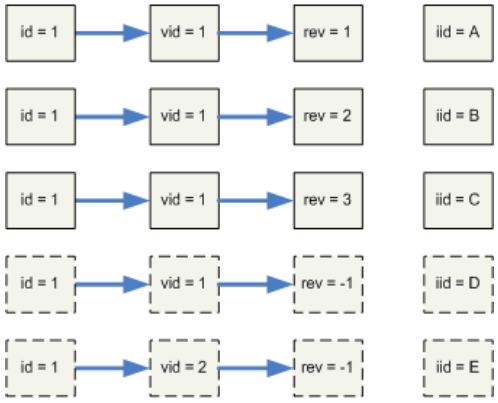

Figure 37: A diagram showing an entity with two drafts

9.3 Linking to Draft Entities

When creating ORBAT data in the service there will often be a mixture of draft entities and published entities used to build up the structure. In order to support this an ORBAT which contains links to draft entities cannot be published since draft entities are not able to be consumed via the general interface. Those draft entities must first be individually published before the ORBAT can be published. An ORBAT uses static links to reference its draft entities, as a dynamic link to any of those entities would get resolved to their published version. These static links can be changed to dynamic links once the draft is published, thus enabling those updates to be reflected in the ORBAT.

9.4 Use Cases

There are a number of use cases for editing an ORBAT which require special business rules to work with the data management process. These use cases are described in more detail below.

9.4.1 Creating an ORBAT from Scratch

There are two variations of this use case, firstly creating a new ORBAT that only references new units which are created at the same time and secondly, creating an ORBAT that references existing units as well as possibly creating new units.

To create a new ORBAT containing new units, which are created at the same time, the client first constructs the ORBAT locally and then puts it into the ORBATDS using the putORBAT operation. This ORBAT will not be immediately available via the general interface, instead the ORBAT and all of its units are put into the *EDITED* state and so will require verification and approval before being published. To retrieve a draft ORBAT saved in this way the client must call the getDraft operation with the `vid` of that entity.

Alternatively, it is possible to create a new ORBAT which references existing units. In this case the client constructs the ORBAT locally again, but for any existing units that it is using it must fetch these from the service. If the client makes any changes to the existing units it must first call the getDraftUnit operation to get a draft of the unit which changes can be made to.

9.4.2 Editing an ORBAT Without Editing its Units

This is the simple editing case where no units are being changed so they don't need to enter the data management process. This use case requires the client to get a draft of the ORBAT for editing by using the getDraftORBAT operation on the administrative interface. A draft based on the latest revision of the ORBAT will be returned to the client. The client can then make the appropriate changes to the ORBAT and store them back in the ORBATDS using the putORBAT operation. This will save the draft version of the ORBAT in the *EDITED* state, adding a metadata entry for the change to that ORBAT. The ORBAT can then be progressed through the data management process.

9.4.3 Editing an ORBAT and its Units

This use case is slightly more complicated as the units being edited also need to enter the data management process. Firstly, the client needs to fetch a draft of the ORBAT to be edited using the getDraftORBAT operation. If the ORBAT contains any draft units, these will also be returned as drafts. When the user wants to edit a unit in the ORBAT, the client must call getDraftUnit to get a draft version of that unit, which should replace any previous instance of it in the client's local data model. Any relationships to the unit should also be changed to static links. Once changes have been made, the entire ORBAT is stored using the putORBAT operation. Any draft units within the ORBAT are put in the *EDITED* state and have new metadata entries added to the list. Any draft ORBATs will also be put in the EDITED state and have a new metadata entry added. Links to drafts are validated to ensure they are static links to the draft instance.

9.4.4 Verifying and Approving an ORBAT

All draft entities are required to go through the different stages of the data management process and eventually be approved for publishing before they become available on the general service interface. In order for an ORBAT to transition into the *APPROVED* state, all entities referenced by the ORBAT must also be in the *APPROVED* state. If the entity being approved is the head of the version timeline, in other words it is the latest version, and a change has been made to the start date of the version then the end date of the previous version must be updated to the start date of the new version being published. This is performed by the service.

9.5 Jurisdiction Based Edit Restrictions

In addition to the data management process, the ORBATDS also supports the partitioning of data according to the owner. Each entity can be assigned a jurisdiction, by default there are four: Army, Navy, Airforce and Joint. These jurisdictions allow data administration to be partitioned in addition to the role(s) a user may have. The implementation restricts editing rights on an entity labelled with a jurisdiction to only those users who belong to that same jurisdiction.

The available jurisdictions can be extended through service configuration. Jurisdictions are simply labels applied to entities and users. These labels can be of any value created by an application administrator of the system when the data service application is deployed. In this way the service supports restricting administration operations on an entity (or group of entities) to a user (or a group of users) based on a shared jurisdiction label.

Entities with no jurisdiction can be edited by any administrative user. Assignment of jurisdiction is done by way of setting the jurisdiction as part of the entity ownership information. Once assigned to an entity, the jurisdiction can only be changed by users who are part of that jurisdiction. Therefore, changing the jurisdiction can only be performed by the same users, by way of removing jurisdiction protection from that entity.

9.6 Data Synchronisation and Multi-Repository Deployment

Multiple instances of the ORBATDS can be deployed on a single network or on multiple networks. Since each data service is supposed to provide a centralised repository of data, multiple instances on the same network need to be managed carefully. In such cases data might need to be synchronised between the various instances. Ideally, each instance is managed so it holds independent data and no data overlap exists. However, there are valid cases where data replication needs to take place. For example, a data service used for deliberate planning could include both contemporary data plus hypothetical data about future forces and their capabilities. In another example, data is best managed on a network that matches the classification of that data (i.e. not on a higher classified network) and so it may be necessary to replicate that data up into data service instances on higher classified networks where it is used.

The ORBATDS supports data synchronisation and multiple repository deployment. Data synchronisation is the process by which data is directly copied from one data repository (origin) and written to another data repository (destination) to ensure that both data sets are kept identical. If data from the origin already exists in the destination repository, this data will be completely removed and overwritten by an updated copy from the origin repository. The main problem created by multiple repository deployments is the potential for data redundancy where a real world entity is managed in multiple repositories, thus creating multiple independent copies of that same entity. This can lead to synchronisation issues, where the entity record is modified inconsistently across the different repositories. Resolving such a conflict automatically is not possible due to the fact that any part of the

conflicting versions across multiple repositories could be right. For example, an entity is updated on service instance A to change its description field while on service instance B its description and weight are changed. It is not possible to automatically determine whether instance A or B or both hold the correct information.

This problem is avoided in the ORBATDS by including a "repositoryId" field on each data entity. This unique identifier represents the "owning" service instance, meaning that only that data service instance is permitted to manage and therefore issue updates of that particular entity. The value of this field is set for the data service by a Super Administrator using an agreed naming convention as part of the configuration of the service at deployment time. The `repositoryId` is different to the unique entity identifier "id" as all entities created or updated by a service instance will share the same `repositoryId` value. When merging with another service instance, this value can be used to determine which entities are managed locally within an instance (and therefore are not to be updated) and which are remotely managed (and therefore could be updated). To prevent inconsistent modification across instances, only local entities (i.e. those that have the same `repositoryId` value as that of the service instance) can be updated by that service. If an administration client tries to modify an entity with a repository id that doesn't match the service, an `IllegalAccessFault` will be thrown.

This approach ensures that no data collisions can occur. Some different deployment examples will now be presented to further demonstrate how the `repositoryID` is used to assist with handling multiple repositories. Figure 38 shows a deployment in a single network environment, the DRN. The majority of data for both the AMDS and ORBATDS will be managed and configured on those DRN data service instances. This DRN production environment is regarded as the source of authoritative data. Any other deployed repositories that need a copy of the entities in that auhoritative repository must have those entities replicated in its repository and are not able to create their own indepenent copy of that data.

The second example, depicted in Figure 39, shows the case of a multiple network environment formed by the DRN and DSN. In this example, the majority of ORBAT and AM data is still being managed on the DRN, while periodically that data is migrated up on to the DSN and merged with the DSN repository. This ensures that the latest DRN managed data is provided to users for planning operations at a higher classification. The DSN `repositoryId` needs to be configured with a unique value so as to partition the data it manages from that it receives from the DRN.

There are two use cases which this setup has to support:

- **Use Case 1:** Users on a higher classified network require data about entities such as the Ready Combat Team that are managed at a lower classification.
- **Use Case 2:** Users on a higher classified network need to modify an entity that exists on a lower classified network with highly classified details pertaining to certain Security Protected Assets (SPA) which cannot exist on the lower network.

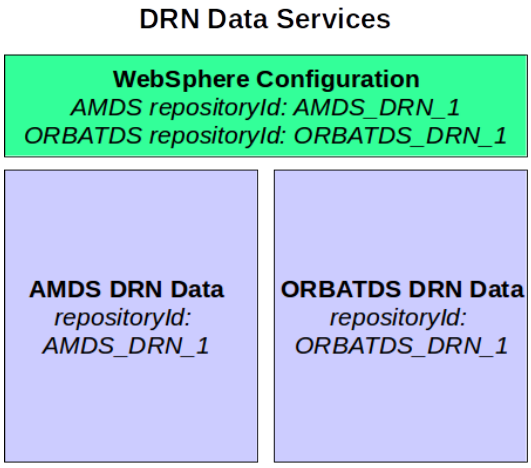

Figure 38: A diagram of a single network environment such as the DRN. The green box indicates the WebSphere service container settings for each service instance. The blue boxes display the repositoryId which each of the entities created in the data repository are assigned.

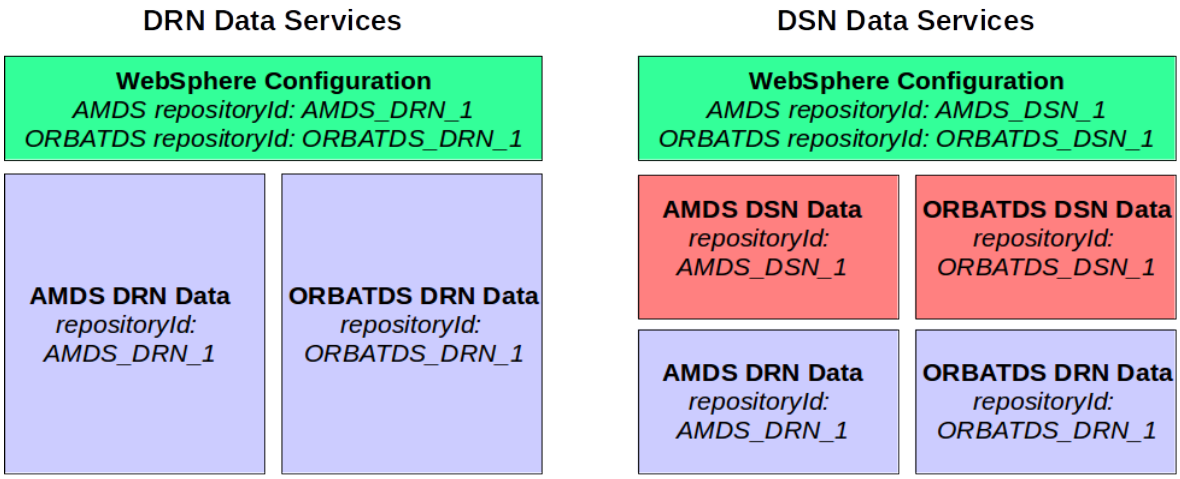

Figure 39: A diagram of a multiple networked environment showing how data is paritioned using repositoryId

Use Case 2 requires an ORBATDS data manager on the DSN to *clone* such entities after they have been updated with a merge from the DRN. The clones can then be updated in the DSN repository. On the DSN the original entities can't be modified as they have the DRN `repositoryId`. However, each cloned entity will be given the `repositoryId` of the service which cloned it, in this case the DSN.

A more complex case is depicted in Figure 40 which involves multiple standalone laptops, each with its own deployed instances of the data services. Each laptop has its own unique `repositoryId` that is applied to any data created or modified on that laptop and the data

repositories on each laptop are initialised with snapshots from an appropriate centrally managed common service instance on a Defence network.

In addition, each Service headquarters (HQ) may require their own data service instances for managing specific raise, train, sustain (RTS) data that is used in exercises etc. In this case each Service HQ manages its own data, but may draw on common authoritative data from that DRN centrally managed instance.

The laptops are easily deployed into theatre and can be used to potentially bring back valuable operational data. Experience has shown that such data needs to be examined and possibly cleansed before it is suitable to merge back into the common data service instance on the appropriate network by virtue of the `repositoryId`.

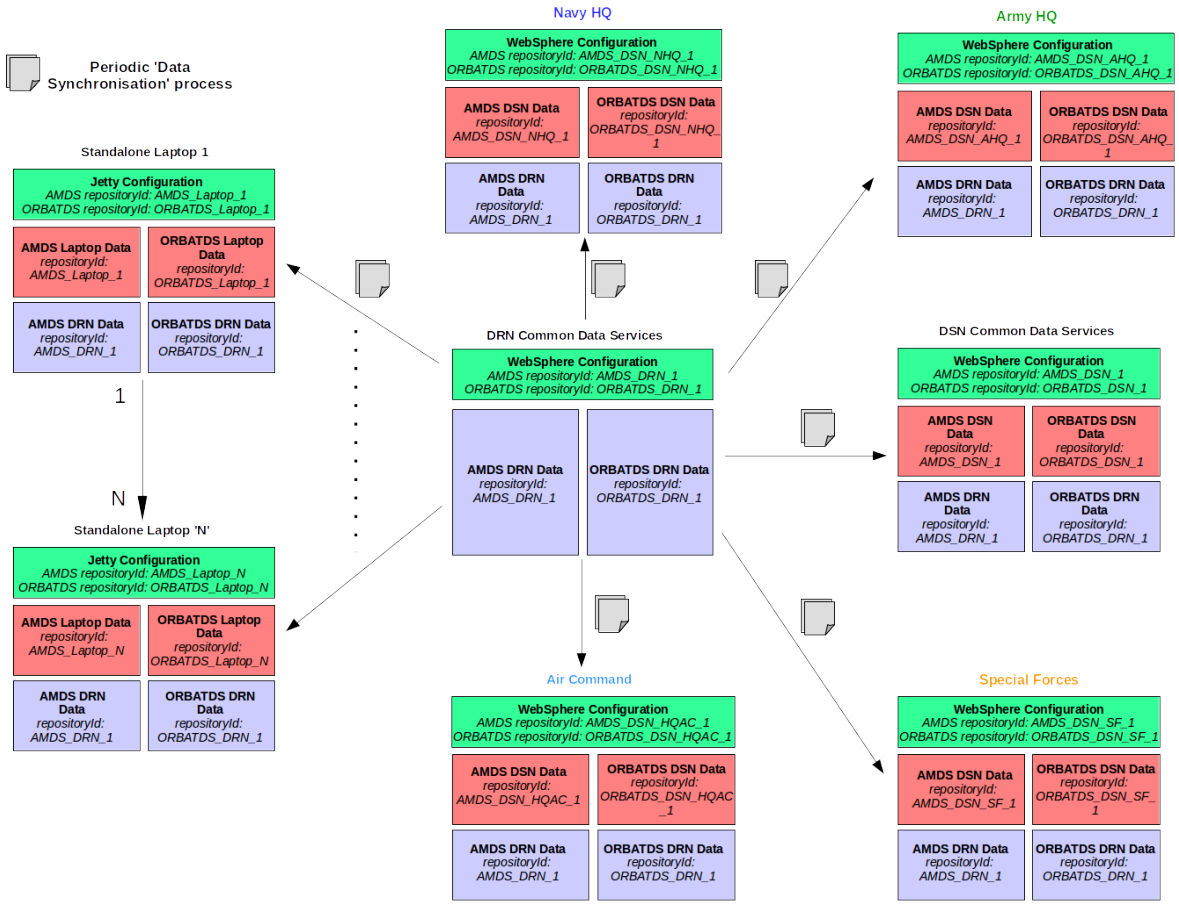

Figure 40: A diagram of a multiple network environment supporting standalone laptops and data service instances for RTS by the services

10. Model Management

10.1 Security

The Java Enterprise Edition (Java EE) web application security architecture is leveraged to implement security in the ORBATDS. Section 3 of the Java EE platform specification document [38] defines the security mechanisms provided. The 'Container Based Security' approach is used whereby the application security requirements are defined as part of its deployment descriptor. When deployed, the enterprise application server hosting the Java EE application, in this case the ORBATDS, provides the security implementation and ensures that the security requirements are met. This approach ensures that a tested and accredited security implementation is furnishing the security needs of the administration client. This is preferred to implementing a custom security model as it guarantees a well-known level of security and provides the benefits of broad industry standard support and interoperability. Thus, authentication, authorisation, confidentiality and integrity are all handled by the enterprise application server upon which the ORBATDS is deployed and can all be configured at deployment time.

At the ORBATDS level there are several user roles requiring different security measures. The ORBATDS has two security domains; a read-only general data access area and a secured area allowing various administrative privileges. The primary purpose of the ORBATDS is to enable client applications (e.g. ViPA) to find and retrieve ORBAT data. This functionality is provided via the General Interface, which allows unrestricted access to all authenticated users of the network on which the ORBATDS is deployed. The classification level of the host network defines the maximum classification of data which can be managed by a given ORBATDS installation. Since all users of that network are assumed to meet the security level requirements, no further access restrictions are necessary. However, since the service is a Java EE application, other security configurations are also supported. This means that the default open read-only access to ORBAT data can be restricted on the basis of specific user groups, if desired.

Write access to ORBAT data via the Administration Interface requires the user to be authenticated beforehand, and depends on their authentication level and security group membership. The user's group memberships determine what functionality is available to that particular user. In order to achieve this with Java EE, the service defines 'security constraints' as part of its deployment descriptor stored in the file web.xml. These security constraints define the different types of user roles, for example 'editor', and specify the resources requiring protection (e.g. the Administration Interface). Together this definition describes what kind of users can gain access to which resources. This is enforced by the application container, as mentioned above. The user details are stored and they are associated with every write on the ORBAT data, thus forming an audit trail. Enough information is stored about the user such that the audit trail persists even in the event that the external user store is no longer available.

The ORBATDS makes use of HTTP basic authentication to securely transport artefacts between the client and the service. JAX-WS supports more robust security schemes, however it was decided not to configure these. HTTP basic authentication does not provide message confidentiality nor integrity checks, which were deemed non-critical since the ORBATDS operates on secure and reliable networks. Transition to a more robust Web Service security mechanism is straight forward as the Java EE environment provides the same language constructs regardless of the mechanism by which the client provides their credentials. Therefore the current HTTP basic authentication mechanism can be substituted for a different mechanism through configuration changes on the client and server.

The above approach defines security constraints independently of individual users or any particular user repository or management scheme. The Java EE application server provides multiple industry standard methods to connect to a user repository. Furthermore, the administration client specific roles can be mapped to a set of users, a user group, a domain etc. The process of defining which users have access is ultimately controlled by the deployment configuration and the external user repository (such as Active Directory).

At the data persistence store level, the security requirement is relatively simple and is provided by the Relational Database Management System (RDBMS) itself. Since the ORBATDS implementation is in effect the only client to the database, only a single system user account with administrative privileges is required. This system user exists entirely within the RDBMS scope and is created specifically for the purpose of maintaining ORBAT data. In the case of a standard ORBATDS deployment, it is an Oracle user with access privileges to the table spaces and schema which store the ORBATDS data. The credentials of this system user are specified in the service configuration and passed to the RDBMS whenever persistent data needs to be accessed. Since Hibernate uses the Java Database Connectivity (JDBC) API, the database connection can be configured with confidentiality and integrity checking through encryption, however the default setup requires only that authentication and authorisation be performed. This is done by verifying that the ORBATDS component provides the correct username and password of the RDBMS system user created for ORBAT persistence.

For more information about the ORBATDS administrative user roles and their privileges please see Section 9.

10.2 Performance

The current implementation of the ORBATDS does not suffer from any known performance issues and appears to be sufficient for current and future needs. However, there are known sources of potential performance problems which may need to be addressed in the future.

The Hibernate object relational mapping framework does incur a performance penalty when processing queries. The framework provides multiple strategies to minimise this overhead, should performance improvements become necessary. Since the ORBATDS data set is relatively static in nature, the caching functionality available in Hibernate could be used to good effect by reducing the response time to return query results. Furthermore, any time-critical queries can be optimised by creating them manually and thus avoiding the framework overhead.

Performance of the ORBATDS is also directly affected by the data store instance used for data persistence, which in the current deployment is mainly Oracle 10g. Techniques for improving Oracle RDBMS performance are well documented by the vendor and many specialised publications [39,40].

The ORBATDS performance can also be improved by simply increasing the memory and processing resources of the machine running the ORBATDS and database, although it's more difficult to address network issues that can slow communication between the ORBATDS and its client applications.

Another source of potential performance problems are poorly specified queries that return far more results than are required by the client. The ORBATDS uses SOAP based web interfaces which require processing time to serialise and de-serialise data objects to and from XML and they also incur network delays when communicating large SOAP messages. Search and get operations that return large numbers of results will necessarily take longer to handle. For these reasons the ORBATDS interfaces provide the ability to limit the number of results returned by a query. Client application developers should take care to safeguard against users specifying poorly constrained queries that can return too much data.

10.3 Concurrency

The business logic in the data service is implemented so that each request and response transaction is executed separately to one another. In this way the data service does not maintain a session and does not need to maintain a state, so it's stateless. However, the HTTP protocol which is used to send and receive SOAP messages does maintain an internal authentication state between a series of requests from the same client. This is an embedded feature of that technology and is hidden from the business logic of the ORBATDS.

In the event that multiple clients try to save the same entity at the same time, the data service implements an optimistic object locking strategy which throws a StaleObjectException to prevent the data from being saved by all except the first client to make this attempt. There is no facility to notify clients when they are concurrently editing the same data because the ORBATDS is stateless and when it returns data to each client, it has no way of knowing if any of those clients will in the future want to save changes to that data. This strategy is sufficient because the data management process partitions the

management of ORBAT entities across a number of jurisdictions such as the military services and joint command, and ORBATs tend not to change that often.

10.4 Data Validation

Data validation is a popular strategy for preventing erroneous data from entering an information system. It is critical to ensure that the data stored in the service data models is valid. Data can be invalid in many ways ranging from syntax errors to fitness for purpose. Identifying low level syntax and data requirements is relatively straight forward.

However, since the data stored in the ORBATDS can be used for many different purposes by many different users, it is difficult to quantify fitness for purpose for all potential applications. Validation can be applied to data in the user-interface (UI), at interfaces with external systems, and between each layer (tier) of the systems architecture. To minimise duplication of human effort and maximise consistency, a validation framework that allows a single defined set of validation rules (constraints) to be applied across multiple system layers is used.

The Hibernate Validator is an implementation of the JSR 303 standard for capturing data object validation rules [7]. This 'Bean Validation' framework effectively provides a means of defining validation rules for Java classes (referred to as 'Beans') which can be expressed in XML files and shared across multiple layers. Figures 41 and 42, taken from the Hibernate Validation documentation [41], show validation being duplicated across multiple layers in contrast to using Hibernate Validator where constraints are defined once in the domain model and shared with each layer. This sharing allows rules to be consistently applied across all layers and eliminates the need for custom implementations of those rules in each layer.

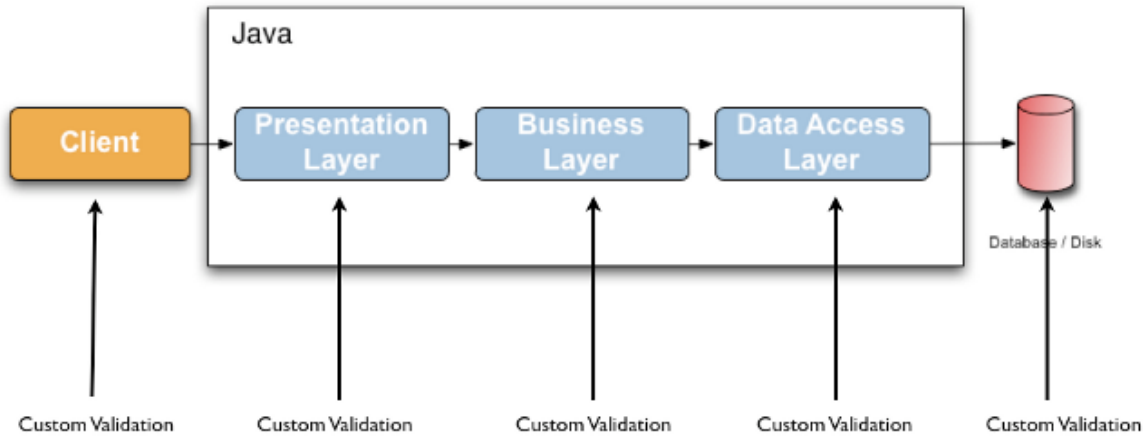

Figure 41: Illustration showing the custom implementation of validation rules across multiple layers

The base layer of validation is provided by the XML Schema definition of the ORBATDS data model. This identifies the bare minimum data validity rules for syntax, data types, value ranges as well as relationship cardinality where appropriate. These rules are applied whenever the data service receives a SOAP-based request from the client.

The validation rules for the ORBATDS are published in an *ORBAT-Validation* software artefact, which can be used by other applications that have a requirement to validate ORBATDS data. The rules check that the ORBAT entities are constructed correctly according to rules imposed by their structure types. An eXtensible Stylesheet Language Transformation (XSLT) is used to generate a human-readable form of the validation rules and can be found in Appendix B.

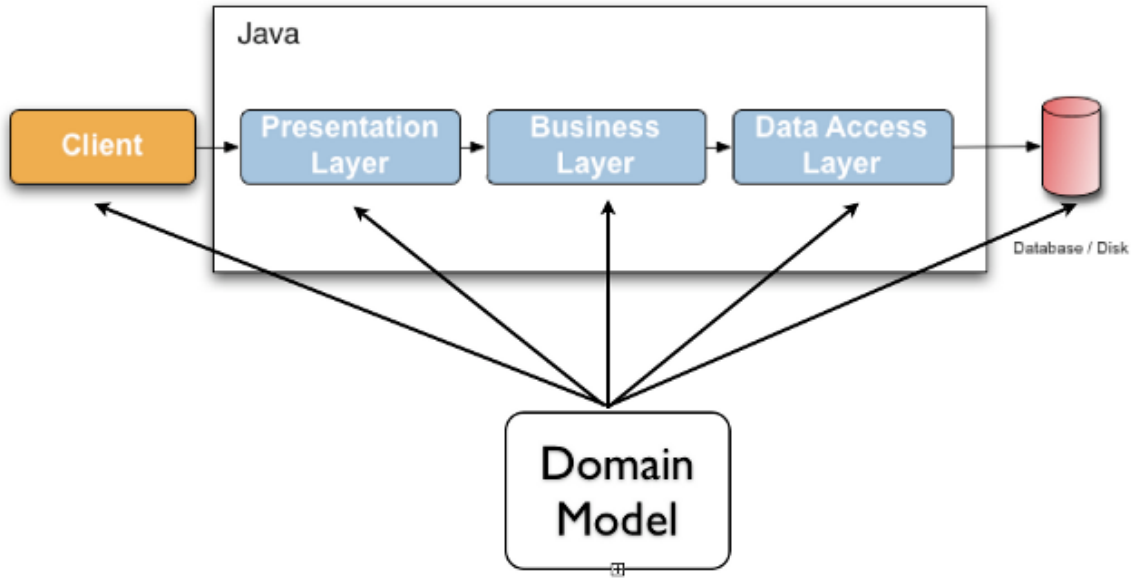

Figure 42: Illustration showing a single set of validation rules defined in the domain model and consistently applied across all layers

Definitions for the various types of checks listed in Appendix B are:

- Linked Unit Containment check: Ensures all the units referenced by links in an ORBAT are contained within the ORBAT's units list.
- Multiple Command check: Ensures that no entity within the link graph has multiple commanding parents.
- Multiple Command Root check: Ensures an ORBAT does not have multiple command roots.
- Cyclic Link check: Ensures the link graph does not contain a cycle.
- Units Have Association Links check: Ensures units in an ORBAT of structure type *OPO* have association links.

- Field Value Combination check: Ensures that a valid field combination has been supplied based on the type of ORBAT. This checks that RelType objects have been constructed correctly based on the type of ORBAT they are being defined for.

10.5 Data Auditing

Data auditing is used to analyse the quality of the ORBATDS data and also to find what changes were made and by whom. The data auditing functionality in the ORBATDS is achieved through the combination of version management, security design and third party tools. Since there is no concept of data deletion in the ORBATDS, only deprecation, data will never be lost. The one exception to this is the deletion of draft objects which haven't been published and so can be deleted. However, data loss due to database failure is still possible, and so a suitable disaster recovery plan must be implemented. Versioning of all data ensures that each and every change made can be reviewed and compared to all prior versions. Furthermore, since the author of each change is recorded with the data, all changes can be reviewed in terms of date, time and person (or system) making those changes.

Data auditing and maintenance tasks can be performed in a number of ways. Any user with access to an ORBATDS can use get operations to perform history retrieval and thus conduct rudimentary auditing tasks on a per data item basis. The ORBATDS Administration Client (ORBATAC) functionality can be used to traverse the version information for any entity, however the ORBATAC does not provide tools to visualise changes between versions. Other tools can be developed to meet specific auditing or maintenance needs.

In addition, the service provides a REST interface which can be used to perform per entity data auditing by traversing its version/revision history. Since the service provides a REST interface, a known entity can be visualised in a web browser. Its history can be traversed giving the user an opportunity to see what changes were made over time, by whom and when. This is a very basic way of performing auditing functions.

The ORBATDS also provides various data management reports that check the data stored in the service. These reports enable data managers to find where the data is inadequate and thus focus their remediation efforts. Using these reports it is also possible to identify data entities which do not meet the data ‘validity’ rules specified by the Hibernate Validator. These reports provide an overview of the quality of data stored in the service with regard to business rules that are defined to support ViPA workbench requirements. A screenshot of the reporting interface can be seen in Figure 43.

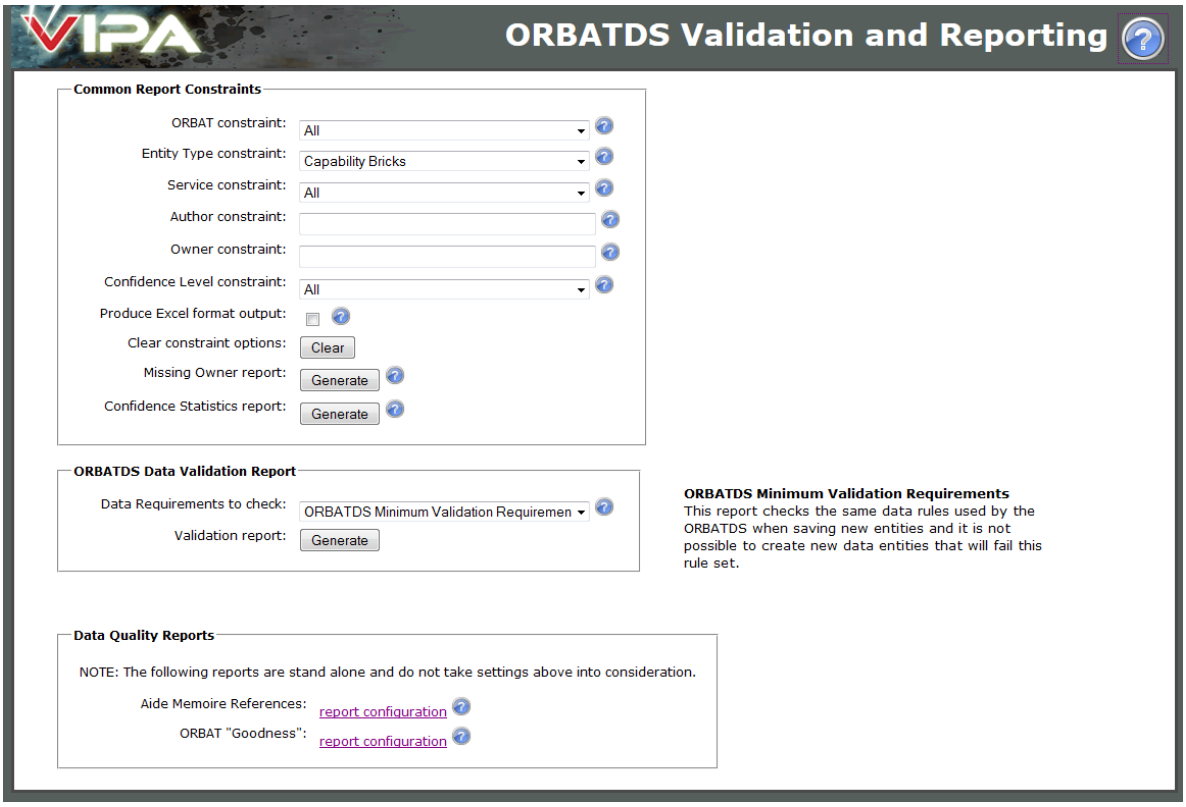

Figure 43: The user interface to the various ORBATDS data management reports

10.6 Data Persistence

The persistence layer must provide responsive retrieval performance and fine grained data access. It should also be possible to search through subcomponents of an ORBAT object graph used by client applications and retrieve only the information required. The requirement for a responsive fine grained search and retrieval system strongly influenced the choice of a commercial relational database. The availability of an enterprise license agreement for Oracle on Defence networks meant it was the logical choice of RDBMS for the ORBATDS. Since ORBATDS data changes slowly over time there are no special strategies required to facilitate transactional performance in the persistence layer. It is expected that after the initial loading of data into an empty database the resultant repository will undergo relatively few changes and updates.

Since a relational database was selected for the data store and the service implementation is object-oriented, an object-relational translation layer was vital. Hibernate ORM [9] was chosen as the persistence framework due to its extensive functionality and support for multiple database platforms. It is also a certified implementation of the JSR-220 [10] JPA specification. The ORBATDS uses Hibernate-specific database mapping files and so does not fully align with the JPA specification.

The Hibernate persistence layer enables the storage and retrieval of Plain Old Java Objects (POJO) instances in a relational database. The Hibernate framework uses a configuration file that specifies the database driver, instance location, references to mapping files and other general configuration settings. The service source code contains two sets of settings, one compatible with standalone deployment and another for Java EE Application server deployment scenarios. Java objects generated by binding the XML schemas are used for data persistence as they provide all the required detail and are POJOs in nature.

The Hibernate configuration file specifies a Java Naming and Directory Interface (JNDI) name for an `org.hibernate.SessionFactory` instance. This SessionFactory instance is retrieved by ORBATDS code when accessing the persistence store. Each Web Service call is in effect an atomic transaction from the service user perspective, and multiple users can call the service simultaneously, each call is isolated to its own `org.hibernate.Session` instance within which a new transaction is created and interaction with the persistent data store is performed via Hibernate APIs.

10.7 Data Mapping

The object-oriented data model used by the ORBATDS is mapped to a relational schema using a fine-grained mapping approach. This results in most objects and relationships of the object model having a corresponding relational schema table. This strategy is favoured over a coarse-grained mapping or direct binary storage due to storage efficiency and fine grain query support. To illustrate, the fine-grained persistence model prevents duplicate data from being stored as data relationships are stored by reference rather than data instance, which ensures data consistency. Furthermore, as each object is mapped to its own table any part of the data model can be queried without the need to traverse parent objects.

Several strategies have been employed to simplify the resultant relational model. For instance, child objects which have a 0..1 cardinality with their parent and do not have children of their own are represented as part of the parent object's table. In the Hibernate mappings this is represented using a *component* construct which has 2..n *property* elements.

Hibernate supports the object-oriented concept of inheritance and provides three methods for mapping to a relational model. These are the single table per class hierarchy strategy, the table per subclass strategy and the table per concrete class strategy. The ORBATDS uses the table per subclass strategy, the result of which is that `DataType` is mapped onto the *datatype* table and a Hibernate *joined-subclass* construct is used to create tables for units and ORBATs. The resultant relational data model is efficient in terms of searching for `DataTypes` and prevents definition duplication for each of the data types.

11. Future Work

This section overviews the main ideas for future enhancements to the ORBATDS that were identified during its development and through its operational use on the DRN, DSN and in theatre.

11.1 Phase Out the Data Management Framework

The data management framework has not been used as originally intended and has in fact degraded the usability of the ORBATAC. The intent was to enable distributed editing of data across jurisdictions, whilst ensuring that only a few subject matter experts could approve data entities for publication and subsequent use by ViPA and other tools.

However, despite a persistent effort over several years to achieve this ideal, it appears that for the foreseeable future the role of managing all the data will continue to rest with a single data administrator, who closely liaises with subject matter experts across the Army, Navy, Air Force, Joint commands and other Defence agencies. Therefore, the data management framework poorly matches the way the data is actually being managed and thus leads to inefficiency, as the single data administrator is forced to act out all the process steps without any tangible benefits to data quality. Unless the kind of distributed data management process originally envisioned through data service requirements is put in place, the software should be modified to better support how the data is actually being managed.

This inefficiency could be alleviated by making a streamlined administration client, which has already been prototyped by DST Group. This customised client allows the top-down creation of ORBATs, so new units can be created at the same time as an ORBAT and moved through the data management process along with that ORBAT. When moving through the states a report is generated for the administrator identifying all the changes made to the ORBAT and its related units since the last save.

Furthermore, since the ORBATDS keeps a log of all changes made to each entity with each change creating a new revision, it is possible to roll back changes to previous versions/revisions to undo incorrect changes. Thus reducing the need for a strict data management framework which could be simplified to a two-phase process, edit and approve. Retaining the draft state of entities is important since editing an entity can span multiple user sessions. Having two steps also enables a second person to independently vet the data entered prior to publication, if desired.

11.2 Data Model Improvements

The ORBATDS has a rather complicated data model with many business rules determining how it is used. This is a consequence of the way that the service was prototyped to demonstrate concepts to users in order to solicit their feedback, identify user

requirements, and gradually consolidating and validating those requirements. The ORBATDS is due for re-factoring of its design to more directly support the use cases in Section 9.4. Furthermore, other improvements have been noted which should improve the usability of the data model and simplify the service and client.

The main issue with the current data model is that the containment of units in ORBATs and ORBATs in ORBATs is represented in different ways. This requires two different methods for dealing with dependencies on the client side. The model and the processing of the model could be greatly simplified by unifying these containment representations.

Further enhancements which should be considered:

- OPO/DPR representation suffers as it tries to reuse existing ORBAT structures, but it's a fundamentally different ORBAT in that it does not contain any structural relationships between entities. An OPO/DPR is simply a flat list of entities with a quantity associated with each. It currently does this by misusing `RelType` objects as described in Section 4.1.4. This could be improved by creating a new subclass of `DataType` for OPOs that is designed to specifically capture the OPO use case and would avoid the overloading of `RelType`. This makes correct usage and validation easier.
- The current data model suffers from some typographic errors which should be corrected.

11.3 Container ORBATs

Some capability bricks can have different configurations of the same ORBAT. For example, there could be a light, medium and heavy configuration for the different equipment load outs. It could be convenient to group those different configurations together to indicate that they are the same capability brick. It could also be useful when assigning a capability brick to a list of capability options that all three configurations are available so the appropriate one can be chosen according to the situation.

This may be implemented by using the `CONTAINER` ORBAT type to allow the grouping of like capability bricks. However, the requirement for the ORBAT to have a command hierarchy would need to be dropped for this ORBAT type.

11.4 Improved Reporting

At present, the structure of the ORBAT data is oriented around the containment relation which is well suited to answer questions about ORBAT composition. However, this is not well suited to answering the kinds of questions a data manager might ask. A data manager needs to sort the data by its various qualities such as: last edit age, designated data manager (in the event that there were multiple), data quality, frequency of use by external tools etc. It is possible to measure some of these data qualities with the current service, but this functionality is not exposed through explicit interfaces.

This kind of functionality would enable more useful data management reports to be generated that identify which data in the ORBATDS fails to meet fit for purpose criteria. This in turn would allow more focussed data remediation efforts to address data gaps, identify poorly performing data owners and identify data entities of very high value to the ADF.

The kind of metadata required to implement the above features would also support the production of data quality trends, which is not currently possible. Trend analysis would help determine if data remediation efforts are on track (e.g. to update key data prior to planning an operation), or whether corrective action is needed.

11.5 Capability Hierarchy

Entities stored in the service can be assigned capabilities from MIL-STD-2525B [31]. The capabilities in that standard are arranged in a hierarchy where there are nodes with the same name at different positions within the hierarchy. Just looking at the capability name doesn't fully describe what it means without also knowing the context provided by its ancestor nodes. For example, there are two capability descriptions with the value "CARGO". It is not until you look at the hierarchy that you see that one refers to a fixed-wing cargo drone aircraft and the other refers to a non-military merchant cargo ship. More of the capability hierarchy should be included in each entity to avoid this confusion.

11.6 Graphical ORBAT Administration Client

The ORBAT Administration Client (ORBATAC) which was initially released with the ORBATDS was found to be sub-optimal in a number of areas. The key shortcomings identified were in 2525 symbol support and editing speed.

The ORBATAC has built in support for core 2525B symbols, but does not support customisation of the core symbols which limits the assignment of icons in the Australian context. The current implementation covers approximately 75% of symbol requirements. By allowing for the customisation of symbols, such as by adding text to them, the implementation would align closer with Australian use. In addition, adopting a later standard such as MIL-STD-2525D [42] would further improve the alignment.

The ORBATAC was designed for use on the outdated Microsoft Internet Explorer 7 web browser. As such, editing features and JavaScript library choice was motivated by browser compatibility and runtime performance, rather than ease of use or to adopt familiar modern web user interface trends. This led the ORBATAC UI to being 'form' driven, where users interact with data predominately using form-based inputs such as text boxes, drop down lists and buttons. This style of input is less intuitive than it could be and limits the kinds of visual presentation and interaction that would significantly aid users' understanding of the data structures they are constructing.

DST Group has prototyped a client which addresses these shortcomings. This ORBAT Builder has the potential to become the première ORBAT Building tool in Defence and can be reused to update the existing ViPA ORBATAC to provide a modern, graphical web-based client to administer the ORBAT data service. It aims to be easy-to-use and to provide artefacts which can feed into the planning process, such as exporting to a ViPA plan. Figure 44 shows a screenshot of the prototype ORBAT Builder which is a significant improvement over the original ORBATAC.

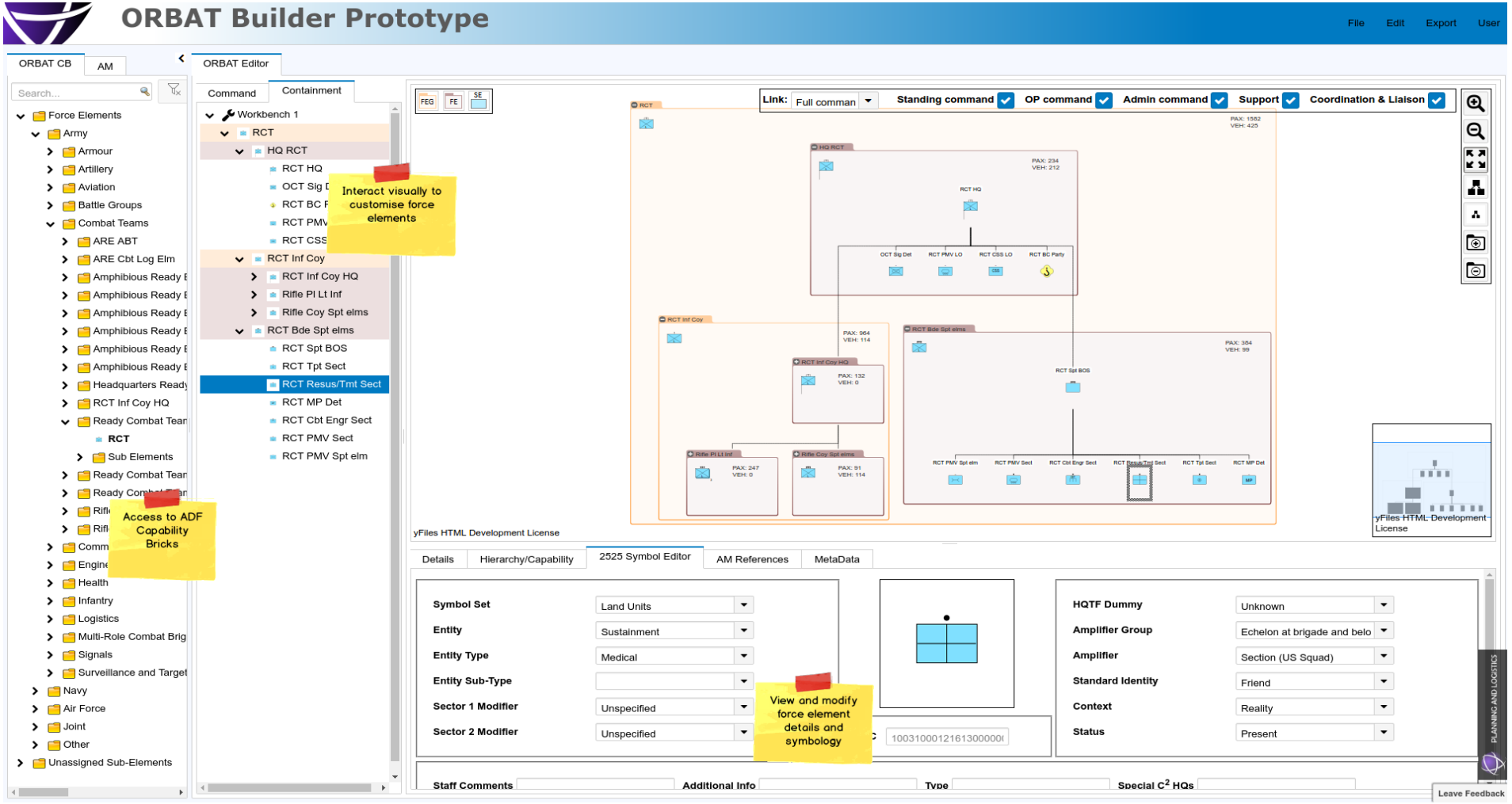

Figure 44: A screenshot of the DST Group developed ORBAT Builder. The leftmost pane contains the explorer which allows the user to find elements in the AMDS and ORBATDS. Next to it is the ORBAT Editor pane which shows the hierarcy of entities in ORBAT that is currently being built. The large pane allows the user to graphically view and edit the ORBAT.

12. References

1. Shannon, B. (2006) *Java Platform, Enterprise Edition (Java EE) Specification, v5.* [Accessed 29 February 2016]; Available from: http://www.oracle.com/technetwork/java/javaee/tech/javaee5-jsp-135162.html

2. Kotamraju, J. (2001) *JSR 224 – Java API for XML-Based Web Services (JAX-WS) 2.0*. [Accessed 29 Februrary 2016]; Available from: http://jcp.org/en/jsr/detail?id=224

3. IBM. *Software – WebSphere Application Server Overview*. [Accessed 29 February 2016]; Available from: http://www-03.ibm.com/software/products/en/was-overview

4. *JAX-WS Reference Implementation—Project Kenai*. [Accessed 29 February 2016]; Available from: https://jax-ws.java.net/

5. *Hibernate*. [Accessed 29 February 2016]; Available from: http://www.hibernate.org/

6. Ort, E. and Mehta, B. (2003) *Java Architecture for XML Binding (JAXB)*. [Accessed 29 February 2016]; Available from: http://www.oracle.com/technetwork/articles/javase/index-140168.html

7. Bernard, E. (2009) *JSR-303 – Bean Validation*. [Accessed 29 February 2016]; Available from: https://jcp.org/en/jsr/detail?id=303

8. *Hibernate Validator.* [Accessed 29 February 2016]; Available from: http://hibernate.org/validator/

9. *Hibernate ORM*. [Accessed 29 February 2016]; Available from: http://hibernate.org/orm/

10. Demichiel, L. and Keith, M. *JSR-220 – Enterprise JavaBeans 3.0*. [Accessed 29 February 2016]; Available from: https://jcp.org/en/jsr/detail?id=220

11. Coomber, G.A. et al. (2005) *ORBAT Services Design Version 1.0.* DSTO-TR-1727, Edinburgh, S.A., Defence Science and Technology Organisation (Australia)

12. Rochkind, M. (1975) *The Source Code Control System*. IEEE transactions on Software Engineering, Vol. SE-1, No. 4, pp. 364-370

13. O'Sullivan B. (2009) *Mercurial: The Definitive Guide*. O'Reilly Media, pp. 11-12.

14. Neumann, P. *(2004) Attaining robust open source software.* Perspectives on Free and Open Source Software, pp. 123-126

15. Surowiecki, K. (2004) *The wisdom of crowds*. Doubleday

16. Collins-Sussman, B. et al (2008) *Version Control with Subversion*. [Accessed 29 February 2016]; Available from: http://svnbook.red-bean.com/en/1.5/svn-book.pdf

17. Wattenberg, M. and Fernanda, V. (2003) *History flow: results*. [Accessed August 2011]; Available from: http://www.research.ibm.com/visual/projects/history_flow/results.htm

18. Commandant Australian Defence Force Warfare Centre (2009) *ADFP 5.0.1: Joint Military Appreciation Process.* Canberra, A.C.T., Defence Publishing Service (Australia). Available from: http://intranet.defence.gov.au/home/documents/data/ADFPUBS/JSPADFP/ADFP5_0_1/FRONT.PDF

19. Director Strategic Logistics Policy, Strategic Logistics Branch, Joint Logistics Group (2006) *Distribution Planning.* ADFP 4.2.2: Distribution Support To Operations. Canberra, A.C.T., Defence Publishing Service (Australia). Available from: http://intranet.defence.gov.au/home/documents/data/ADFPUBS/JSPADFP/ADFP4_2_2/03A.pdf

20. Bartel, D (CSC Australia) (2014) *ViPA 2.0 System Specification (SS)*, Release 4.0.

21. Gamma, E. et al. (1995) *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts, United States of America, Addison-Wesley

22. *Australian Defence Glossary*. [Accessed 9 February 2015]; Available from: http://adg.eas.defence.mil.au/

23. Chambers, S. and Freeman, J. (2014) *Cloud Terrain Generation and Visualization Using Open Geospatial Standards.* In: Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC), [Accessed 29 February 2016]; Available from: http://external.opengeospatial.org/twiki_public/pub/CommonDataBaseSWG/WebHome/14308_final_CAE_J7_paper.pdf

24. United States Joint Forces Command (USJFCOM) (2010) *Joint Live Virtual and Constructive (JLVC) Federation Integration Guide.* Norfolk, Virginia, U.S. [Accessed 29 February 2016]; Available from: http://www.dtic.mil/dtic/tr/fulltext/u2/a521311.pdf

25. AVM C. Hingston et. al. (2003) *An Evaluation of ADF Logistics Support to Operations in the Middle East With a View to Informing Future Logistic Capability Development*.

26. Thuve, H. (2002) *TOPFAS (Tool for Operational Planning, Force Activation and Simulation)*. In: 6th International Command and Control Research and Technology Symposium

27. Tamai, S. (2009) *TOPFAS Tools for Operational Planning Functional Area Service: what is this?* NRDC-ITA Magazine, Issue 14, pp 20-22

28. Department of Defence (2010) *Single Information Environment (SIE) Architectural Intent*.

29. *TOPFAS Web Flyer* (2010) [Accessed 29 February 2016]; Available from: https://www.eiseverywhere.com/file_uploads/9391b19efc1518a9ac4e685ce9420f3a_TOPFAS_Web_Flyer_Aug_2010.pdf

30. VCDF Directive 02/2003 (2003) to MAJGEN P.F.Haddad

31. U.S. Department of Defence (2007) *Department of Defence Interface Standard – Common Warfighting Symbology, MIL-STD-2525B with CHANGE 2*

32. W3C. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).* [Accessed 14 July 2015]; Available from: http://www.w3.org/TR/soap12/

33. W3C. *Web Services Description Language (WSDL) 1.1.* [Accessed 14 July 2015]; Available from: http://www.w3.org/TR/wsdl

34. *Management of Logistics Data to Support the Planning of Australian Defence Force Operations*, DEFLOGMAN, Part 2, Volume 8. [Accessed 22 February 2016]; Available from: http://intranet.defence.gov.au/home/documents/data/DEFPUBS/DEPTMAN/DEFLOGMAN/VOLUME8/10.pdf

35. JPG Partners (2012). *ViPA Data Services Management Framework (VDSMF)*, version 5.0.

36. *Jetty – Servlet Engine and HTTP Server*. [Accessed 29 February 2016]; Available from: http://www.eclipse.org/jetty/

37. *HyperSQL DataBase*. [Accessed 29 February 2016]; Available from: http://hsqldb.org/

38. Shannon, B. (2006) *Java Platform, Enterprise Edition (Java EE) Specification v5*, Sun Microsystems

39. Chan, I. (2008) *Database Performance Tuning Guide*. [Accessed 29 February 2016]; Available from: http://docs.oracle.com/cd/B19306_01/server.102/b14211/toc.htm

40. Niemiec, R. (2007) *Oracle Database 10g Performance Tuning Tips & Techniques*. McGraw-Hill Eduction

41. *Hibernate Validation Preface*. [Accessed 7 February 2016]; Available from: http://docs.jboss.org/hibernate/validator/4.3/reference/en-US/html/preface.html

42. U.S. Department of Defence (2014) *Department of Defence Interface Standard – Common Warfighting Symbology.* MIL-STD-2525D

43. Schaffner, B. (2010) *World Climates*. [Accessed 22 March 2016]; Available from: http://www.blueplanetbiomes.org/climate.htm

44. Command of the Defence Council (1986) *APP-6 Military Symbols for Land Based Systems.* NATO Military Standardization and Terminology

45. Director Doctrine, Australian Defence Force Warfare Centre (1992) *Australian Defence Force Publication, Staff Duties Series, ADFP 103, Abbreviations and Military Symbols.* Defence Centre, Canberra

46. Object Managment Group (2005) *UML Infrastructure Specification, v2.0.* [Accessed 1 April 2016]; Available from: http://www.omg.org/spec/UML/2.0/

47. National Geospatial-Intelligence Agency (2010) *Geopolitical Entities and Codes (Formerly Federal Information Processing Standards Publication 10-4: Countries, Dependencies, Areas of Special Sovereignty, and Their Principal Administrative Divisions).* Available From: http://geonames.nga.mil/gns/html/PDFDocs/GEOPOLITICAL_CODES.pdf

48. Fallside, D. and Walmsley, P. (2004) *XML Schema Part 0: Primer Second Edition.* [Accessed 11 April 2016]; Available from https://www.w3.org/TR/xmlschema-0/

13. Appendix A: ORBAT Data Model Schema

This is an W3C XML Schema version 1.0 [48] which defines the ORBAT Data Model.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns="http://orbat.vipa.dsto"
xmlns:orb="http://orbat.vipa.dsto"
xmlns:am="http://am.dsto"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
xmlns:xjc="http://java.sun.com/xml/ns/jaxb/xjc"
xmlns:geo="http://geo.vipa.dsto"
xmlns:com="http://commons.vipa.dsto"
targetNamespace="http://orbat.vipa.dsto"
elementFormDefault="qualified"
attributeFormDefault="unqualified"
version="0.1"
jaxb:version="2.1"
jaxb:extensionBindingPrefixes="xjc">
  <xs:import namespace="http://am.dsto" schemaLocation="./AM.xsd"/>
              <xs:import       namespace="http://commons.vipa.dsto"
schemaLocation="./VIPAcommons.xsd"/>
                <xs:import        namespace="http://geo.vipa.dsto"
schemaLocation="./VIPAgeo.xsd"/>
  <xs:complexType name="DataType">
    <xs:annotation>
        <xs:documentation>An object representation which captures
common elements between ORBATs and Units</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="name" type="xs:string">
        <xs:annotation>
            <xs:documentation>Name of this Entity which the users
have developed</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="formalName" type="xs:string">
        <xs:annotation>
                        <xs:documentation>The  2525B  Symbol  Name
description.</xs:documentation>
        </xs:annotation>
      </xs:element>
                <xs:element  name="description"  type="xs:string"
minOccurs="0">
        <xs:annotation>
                      <xs:documentation>Description  given  to  the
ORBAT</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="timeType" type="com:QTimeType"/>
```

```
      <xs:element name="primaryCapability" type="xs:string">
        <xs:annotation>
            <xs:documentation>Defines the capability of the ORBAT.
What the ORBAT is capable of providing and delivering. The
capabilities are tied to the battle dimension. </xs:documentation>
        </xs:annotation>
      </xs:element>
       <xs:element name="secondaryCapability" type="CapabilityType"
minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
           <xs:documentation>Defines the secondary capabilities of
the ORBAT. What the ORBAT is capable of providing and delivering.
The capabilities are tied to the battle dimension. Currently, only
ground capability is supported.</xs:documentation>
        </xs:annotation>
      </xs:element>
                              <xs:element    name="battleDimension"
type="com:battleDimensionEnum">
        <xs:annotation>
               <xs:documentation>The Battle dimension defines the
primary mission area for the war-fighting entity within the battle-
space. If the battle dimension cannot be or has not been
determined, it is considered to be unknown. If the battle dimension
is known, an object can have a mission area above the earth's
surface (i.e., in the air or outer space), on the earth's surface,
or below the earth's surface. If the mission area of an object is
on the earth's surface, it can be either on land or sea. The ground
dimension includes those mission areas on the land surface and is
divided into units, equipment, and installations. The sea surface
dimension includes those objects whose mission area is on the sea
surface, whereas the subsurface dimension includes objects whose
mission area is below the sea surface. More information can be seen
in the MIL-STD-2525B  standard.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="echelon" type="com:forceSizeEnum">
        <xs:annotation>
          <xs:documentation>The echelon the commanding level of the
entity. i.e. Squad, Section, Platoon, etc.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="affiliation" type="com:affiliationEnum">
        <xs:annotation>
           <xs:documentation>Affiliation refers to the threat posed
by the warfighting object being represented. The basic affiliation
categories are unknown, friend, neutral, and hostile. Using the
APP-6A NATO specification.</xs:documentation>
        </xs:annotation>
      </xs:element>
           <xs:element  name="environmental"  type="am:climateEnum"
minOccurs="0">
```

```
        <xs:annotation>
          <xs:documentation>Describes the climatic conditions where
the entity is physically located. Enumeration of climate types (use
Keoppen's climate classification
http://www.blueplanetbiomes.org/climate.htm?).</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="owner" type="am:AuthorType" minOccurs="0"/>
      <xs:element name="symbol2525B" type="xs:string">
        <xs:annotation>
            <xs:documentation>A symbol ID code is a 15-character
alphanumeric identifier that provides the information necessary to
display or transmit a tactical symbol between MIL-STD-2525
compliant systems. Position 1, coding scheme, indicates which
overall symbology set a symbol belongs to. Position 2, affiliation,
indicates the symbol's affiliation. Position 3, battle dimension,
indicates the symbol's battle dimension. Position 4, status,
indicates the symbol's planned or present status. Positions 5
through 10, function ID, identifies a symbol's function. Each
position indicates an increasing level of detail and
specialisation. Positions 11 and 12, symbol modifier indicator,
identify indicators present on the symbol such as echelon,
feint/dummy, installation, task force, headquarters staff, and
equipment mobility. Positions 13 and 14, country code, identifies
the country with which a symbol is associated. Country code
identifiers are listed in the Federal Information Processing
Standard (FIPS) Pub 10 series. Position 15, order of battle,
provides additional information about the role of a symbol
in the battlespace. More information can be seen in the MIL-STD-
2525B standard.</xs:documentation>
        </xs:annotation>
      </xs:element>
           <xs:element name="attributes" type="am:AttributeType"
minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
              <xs:documentation>List of attributes of this orbat
represented using tripple, rather than name-value pair notation to
allow for proper specification of name context.</xs:documentation>
        </xs:annotation>
      </xs:element>
                  <xs:element ref="am:metaData" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="role" type="xs:string" minOccurs="0">
        <xs:annotation>
           <xs:documentation>Role is something that is tied to the
specific force elements (based on ASJETS 'proficiencies') and is
used when constructing OPOs where the prep desk officer might seek
to search for FEs that meet the requirements to fulfil a specific
role.</xs:documentation>
        </xs:annotation>
      </xs:element>
```

```
                <xs:element   name="service"   type="com:serviceEnum"
minOccurs="0">
        <xs:annotation>
          <xs:documentation>Defines the service which the entity is
associated, i.e. Army, Navy, Air Force.</xs:documentation>
        </xs:annotation>
      </xs:element>
           <xs:element  name="sourceData"  type="am:SourceDataType"
minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string">
      <xs:annotation>
           <xs:documentation>An identifier for the entity object
represented. Used in conjunction with vid and rev attributes to
identify a unique instance of an entity object. Each unique
instance (combination of id, vid and rev values) will also have a
unique iid value to simplify data management.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="iid" type="xs:ID" use="required">
      <xs:annotation>
          <xs:documentation>This is the instance identifier which
uniquely defines each individual entity instance and is also the
parameter used as the primary key for storing the entity objects
inside the database.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="vid" type="xs:string">
      <xs:annotation>
        <xs:documentation>Version id for this entity. Each version
of a entitymay have 0-n revisions identified by rev attribute.  A
new version is created to represent substantial changes in the
entity which the entity identified by id attribute represents.
Since an entity exists on the temporal scale, versions represent
valid points/periods on the timeline. vid values must uniquely
identify a particular version history of an entity, i.e. must not
be reused with instances that have a different id
value.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="rev" type="xs:int">
      <xs:annotation>
        <xs:documentation>Represents a revision sequence number for
the version history identified by vid attribute. Revisions are used
to manage records of changes which do not constitute a new version
of an entity, for instance when fixing a typo in an entity which is
otherwise valid.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="isHead" type="xs:boolean">
      <xs:annotation>
```

```
        <xs:documentation>Identifies whether this instance of the
entity is the most recent revision of a particular
version.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="security" type="xs:string">
      <xs:annotation>
           <xs:documentation>Describes the security level of the
entity described.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="prevId" type="xs:string">
      <xs:annotation>
                <xs:documentation>Version Id of previous entity
object.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="nextId">
      <xs:annotation>
                  <xs:documentation>Version Id of next entity
object.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="state" type="am:state">
      <xs:annotation>
                   <xs:documentation>The current state of the
object.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="lockVersion" type="xs:int">
      <xs:annotation>
        <xs:documentation>Used for optimistic locking when editing
draft versions of the objects.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="repositoryId" type="xs:string">
      <xs:annotation>
         <xs:documentation>This identifies the repository in which
this object was created. It can be used for merging and data
management.  Clients should not set or change this property when
creating or updating the object, if they do the changes will be
ignored by an AMDS service which will set the repositoryId to it's
own id value. An ORBAT service should not allow updates to object
with repositoryIds which are different to it's
id.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="lazy" type="xs:boolean">
      <xs:annotation>
```

```
          <xs:documentation>Specifies whether the entity is a cut
down version of the full entity definition, i.e. a lazy/stub
representation</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="ORBATDataType">
    <xs:annotation>
         <xs:documentation>An object representation which defines
ORBAT specific elements</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="DataType">
        <xs:sequence>
          <xs:element name="nationality" type="com:nationalityEnum"
minOccurs="0">
            <xs:annotation>
                 <xs:documentation>Defines the nationality of the
ORBAT. NATO standard two letter County Codes. This nationality code
is used inside the MIL-STD-2525B standard and inside
_DataType.symbol2525B_ parameter representation.</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="noUnits" type="xs:int" minOccurs="0">
            <xs:annotation>
                    <xs:documentation>Defines the number of units
associated with the ORBAT</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="noLinks" type="xs:int" minOccurs="0">
            <xs:annotation>
                    <xs:documentation>Defines the number of links
associated with the ORBAT</xs:documentation>
            </xs:annotation>
          </xs:element>
                                  <xs:element   name="structureType"
type="StructureTypeEnum">
            <xs:annotation>
                <xs:documentation>Defines the overall strcture and
purpose of the entity defines, i.e. Operational Planning Objective
(OPO), Unit Entitlement (UE).</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="ORBATType">
    <xs:annotation>
      <xs:documentation>An ORBAT definition comprising of a list of
unit nodes and relationships. Multiple relationship structures can
```

```
exist by vertue of typing each relationship map. Relationship
structures are not restricted.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="ORBATDataType">
        <xs:sequence>
            <xs:element name="units" type="UnitType" minOccurs="0"
maxOccurs="unbounded">
            <xs:annotation>
                        <xs:documentation>Stores a list of unit
entities.</xs:documentation>
            </xs:annotation>
          </xs:element>
             <xs:element name="links" type="RelType" minOccurs="0"
maxOccurs="unbounded">
            <xs:annotation>
                  <xs:documentation>Stores a list of relationship
entities.</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="ORBATTypeStub">
    <xs:annotation>
       <xs:documentation>An ORBAT which defines a cut down version
of the full ORBAT without the associated units or
links</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="ORBATDataType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="UnitDataType">
    <xs:annotation>
      <xs:documentation>An object representation which defines Unit
specific elements</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="DataType">
        <xs:sequence>
                    <xs:element name="noAmObjRef" type="xs:int"
minOccurs="0">
            <xs:annotation>
                  <xs:documentation>Specifies the number of Aide
Memoire Object references associated with the
Unit</xs:documentation>
            </xs:annotation>
          </xs:element>
```

```
                <xs:element name="position" type="geo:PositionType"
minOccurs="0">
            <xs:annotation>
                  <xs:documentation>Represents a unique geospatial
position where the entity is located.</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="type" type="TypeEnum">
            <xs:annotation>
                <xs:documentation>Defines the type of entity, i.e.
Capability Brick (type) or Instance.</xs:documentation>
            </xs:annotation>
          </xs:element>
                                <xs:element   name="maintenanceType"
type="MaintenanceTypeEnum" minOccurs="0"/>
            <xs:element name="wuc" type="am:weaponUserCategoryEnum"
minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="UnitType">
    <xs:annotation>
           <xs:documentation>A structural representation of the
functional ORBAT node</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="UnitDataType">
        <xs:sequence>
              <xs:element name="amObjRefs" type="UnitTypeAmObjRef"
minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
                  <xs:documentation>Stores a list of Aide Memoire
references, Equipment, Supply Items, etc.</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="UnitTypeStub">
    <xs:annotation>
      <xs:documentation>A Unit which defines a cut down version of
the full Unit without the associated Aide Memoire objects
attached</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="UnitDataType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="RelType">
```

```
    <xs:annotation>
      <xs:documentation>A parent child relationship between either
two ORBAT or Unit entities.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
            <xs:element  name="attributes"  type="am:AttributeType"
minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
               <xs:documentation>List  of  attributes  of  this  link
represented using tripple, rather than name-value pair notation to
allow for proper specification of name context.</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID">
      <xs:annotation>
        <xs:documentation>Relationship GUID</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="name" type="xs:string">
      <xs:annotation>
        <xs:documentation>Relationship name/label that may be shown
to the user.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
          <xs:attribute   name="type"   type="com:relationshipType"
use="required">
      <xs:annotation>
        <xs:documentation>Relationship type, i.e. Command, Support,
etc</xs:documentation>
      </xs:annotation>
    </xs:attribute>
           <xs:attribute   name="parentOrbatVid"   type="xs:string"
use="optional">
      <xs:annotation>
         <xs:documentation>Parent ORBAT object version ID. Use this
identifier   to   specify   a   soft   link   to   an   orbat
version.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
           <xs:attribute   name="parentOrbatIid"   type="xs:string"
use="optional">
      <xs:annotation>
               <xs:documentation>The  ORBAT  instance  which  this
Relationship was intended. Effectively creates a hard link to a
particular instance.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
           <xs:attribute   name="childOrbatVid"   type="xs:string"
use="optional">
      <xs:annotation>
```

```
         <xs:documentation>Child ORBAT object version ID. Use this
identifier    to    specify    a    soft    link    to    an    orbat
version.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
            <xs:attribute   name="childOrbatIid"   type="xs:string"
use="optional">
      <xs:annotation>
               <xs:documentation>The  ORBATs  version  which  this
Relationship was intended. Effectively creates a hard link to a
particular instance.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
           <xs:attribute   name="parentUnitVid"   type="xs:string"
use="optional">
      <xs:annotation>
                        <xs:documentation>Parent   unit   version
Id.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
            <xs:attribute   name="parentUnitIid"   type="xs:IDREF"
use="optional">
      <xs:annotation>
         <xs:documentation>The Parents iid which this Relationship
was intended.</xs:documentation>
        <xs:appinfo>
          <jaxb:property>
            <jaxb:baseType name="UnitType"/>
          </jaxb:property>
        </xs:appinfo>
      </xs:annotation>
    </xs:attribute>
            <xs:attribute   name="childUnitVid"   type="xs:string"
use="optional">
      <xs:annotation>
                       <xs:documentation>Child   UnitType   object
vid.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
             <xs:attribute   name="childUnitIid"   type="xs:IDREF"
use="optional">
      <xs:annotation>
           <xs:documentation>The child Unit's iid to form a hard
link.</xs:documentation>
        <xs:appinfo>
          <jaxb:property>
            <jaxb:baseType name="UnitType"/>
          </jaxb:property>
        </xs:appinfo>
      </xs:annotation>
    </xs:attribute>
```

```
    <xs:attribute name="cardMin" type="xs:int" default="0">
      <xs:annotation>
        <xs:documentation>Minimum cardinality of the relationship.
Default value of 0.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="cardMax" type="xs:int" default="1">
      <xs:annotation>
        <xs:documentation>Maximum cardinality of the relationship.
Default value of 1.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="UnitTypeAmObjRef">
    <xs:annotation>
            <xs:documentation>Represents a List of AM object
references.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="type" type="am:amObjectTypeEnum">
        <xs:annotation>
                  <xs:documentation>The AM object type being
represented.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="maximum" type="AmObjMRefType">
        <xs:annotation>
             <xs:documentation>Refers to the Operational Level of
Capability, OLOC. OLOC is the task-specific level of capability
required by a force to execute its role in an operation at an
acceptable level of risk.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="minimum" type="AmObjMRefType">
        <xs:annotation>
                <xs:documentation>Refers to the Minimum Level of
Capability, MLOC. MLOC is the lowest level of capability from which
a Unit can achieve its operational level of capability (OLOC)
within readiness notice (RN), and it encompasses the maintenance of
core skills, safety and professional standards. It is a term
employed only within the six OE.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="required" type="AmObjMRefType">
        <xs:annotation>
              <xs:documentation>Refers to the level of equipment
necessary to enable a unit to conduct routine training and
administrative activities. Know as the Full Time Entitlement,
FTE</xs:documentation>
        </xs:annotation>
      </xs:element>
```

```
        <xs:element name="amRef" type="AmObjMRefType" minOccurs="0"
maxOccurs="unbounded">
         <xs:annotation>
                  <xs:documentation>Represents an Unit Entitlement
equipment assignment to the Unit entity. Specifies the Loan
Entitlement (LE), and any additional forms of
capability.</xs:documentation>
         </xs:annotation>
       </xs:element>
           <xs:element name="wuc" type="am:weaponUserCategoryEnum"
minOccurs="0">
         <xs:annotation>
           <xs:documentation>Specifies the Weapon User Category for
this Aide memoire allocated item.</xs:documentation>
         </xs:annotation>
       </xs:element>
     </xs:sequence>
     <xs:attribute name="id" type="xs:string">
       <xs:annotation>
          <xs:documentation>The persisted class idnetifier (primary
key)</xs:documentation>
       </xs:annotation>
     </xs:attribute>
     <xs:attribute name="amid" type="xs:string" use="required">
       <xs:annotation>
                        <xs:documentation>Aide Memoire reference
identifier.</xs:documentation>
       </xs:annotation>
     </xs:attribute>
     <xs:attribute name="name" type="xs:string" use="required">
       <xs:annotation>
           <xs:documentation>Represents the Name of AM Object when
first assigned. Loosely couples the AMDS and the ORBATDS.
</xs:documentation>
       </xs:annotation>
     </xs:attribute>
     <xs:attribute name="version" type="xs:int" use="required">
       <xs:annotation>
            <xs:documentation>The version of the AM reference when
first assigned.</xs:documentation>
       </xs:annotation>
     </xs:attribute>
     <xs:attribute name="prescribed" type="xs:boolean">
       <xs:annotation>
          <xs:documentation>If true, it indicates that the unit has
additional specialist capability for that piece of equipment and
can therefore perform all levels on maintenance on that piece of
equipment, i.e. Close (1st Line)/Integral (2nd Line)/General
(3rd/4th Line). If false the Unit can not perform all levels of
maintenance.</xs:documentation>
       </xs:annotation>
```

```
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="AmObjMRefType">
    <xs:annotation>
        <xs:documentation>Defines a form of capability toward the
assigned unit.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="capabilityType" type="xs:string"/>
      <xs:element name="multiplicity" type="xs:double"/>
    </xs:sequence>
    <xs:attribute name="id"/>
  </xs:complexType>
  <xs:complexType name="CapabilityType">
    <xs:annotation>
       <xs:documentation>Pairs a capability name with the MIL-STD-
2525 symbol code representing that capability.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="capabilityName" type="xs:string">
        <xs:annotation>
                        <xs:documentation>A  string  describing  a
capability.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="symbolCode" type="xs:string">
        <xs:annotation>
            <xs:documentation>This is the MIL-STD-2525 symbol code
which   encapsulates   the   capability   described   in   the
capabilityName.</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="optional"/>
  </xs:complexType>
  <xs:simpleType name="TypeEnum">
    <xs:annotation>
          <xs:documentation>The  type  of  entity  which  is  being
defined</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Brick"/>
      <xs:enumeration value="Instance"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="StructureTypeEnum">
    <xs:annotation>
        <xs:documentation>Describes the purpose of the structure
which the entity is defined, i.e. UE, OPO ORBATs</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
```

```
      <xs:enumeration value="UE"/>
      <xs:enumeration value="OO"/>
      <xs:enumeration value="OPO"/>
      <xs:enumeration value="OO_BRICK"/>
      <xs:enumeration value="OU_BRICK"/>
      <xs:enumeration value="CONTAINER"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="MaintenanceTypeEnum">
    <xs:annotation>
       <xs:documentation>Described the type of maintenance which a
force element is capable of providing.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Integral"/>
      <xs:enumeration value="Close"/>
      <xs:enumeration value="General"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

14. Appendix B: ORBAT Data Model Validation Rules

This appendix lists the validation rules which are defined for the ORBATDS. For each of the classes in the ORBAT Data Model it lists constraints which apply to a specific field within that class and constraints which are class-wide and so can involve multiple fields or cross class boundaries. More information about data validation within the ORBATDS can be found in Section 10.4.

Some constraints are noted as having a specific validation group. These constraints are only checked in specific circumstances, such as at persistence, and are not general constraints on the class.

- Class: dsto.vipa.orbat.AmObjMRefType
  - Constraints specified per field:
    - capabilityType
      - Must not be null (empty).
      - Size must have minimum value of 1 and maximum value of 100.
    - multiplicity
      - Minimum value is 0.
- Class: dsto.am.AttributeType
  - Constraints specified per field:
    - name
      - Must not be null (empty).
      - Size must have minimum value of 1 and maximum value of 100.
    - value
      - Must not be null (empty).
      - Size must have minimum value of 1 and maximum value of 1000.
    - vocabulary
      - Size must have minimum value of 1 and maximum value of 100.
- Class: dsto.am.AuthorType
  - Constraints specified per field:
    - id
      - Size must have minimum value of 1 and maximum value of 255.
    - displayName
      - Size must have minimum value of 1 and maximum value of 400.
    - username
      - Size must have minimum value of 1 and maximum value of 200.
    - email
      - Field must be a valid email address.
      - Size must have minimum value of 1 and maximum value of 400.
    - sourceURI
      - Size must have minimum value of 1 and maximum value of 400.
    - jurisdiction
      - Size must have minimum value of 1 and maximum value of 200.
- Class: dsto.vipa.orbat.CapabilityType

    - Constraints specified per field:
        - capabilityName
            - Must not be null (empty).
            - Size must have minimum value of 1 and maximum value of 160.
        - symbolCode
            - Size must have minimum value of 0 and maximum value of 15.
- Class: dsto.vipa.orbat.DataType
    - Constraints specified per field:
        - iid
            - Must not be null (empty).
            - Size must have minimum value of 1 and maximum value of 255.
        - vid
            - Must not be null (empty).
            - Size must have minimum value of 1 and maximum value of 255.
        - id
            - Must not be null (empty). **Only when the following validation group(s) are being checked:** dsto.validation.Persistence,
            - Size must have minimum value of 1 and maximum value of 255.
        - nextId
            - Size must have minimum value of 1 and maximum value of 255.
        - prevId
            - Size must have minimum value of 1 and maximum value of 255.
        - security
            - Size must have minimum value of 1 and maximum value of 255.
        - name
            - Must not be null (empty).
            - Size must have minimum value of 1 and maximum value of 100.
        - formalName
            - Must not be null (empty).
            - Size must have minimum value of 1 and maximum value of 255.
        - description
            - Size must have minimum value of 0 and maximum value of 3000.
        - timeType
            - Must not be null (empty).
            - The referenced object must be valid.
        - primaryCapability
            - Must not be null (empty).
            - Size must have minimum value of 1 and maximum value of 100.
        - secondaryCapability
            - The referenced object must be valid.
        - battleDimension
            - Must not be null (empty).
        - echelon
            - Must not be null (empty).

        - affiliation
            - Must not be null (empty).
        - service
            - Must not be null (empty).
        - symbol2525B
            - Size must have minimum value of 1 and maximum value of 255.
        - role
            - Size must have minimum value of 1 and maximum value of 1500.
        - attributes
            - The referenced object must be valid.
        - owner
            - The referenced object must be valid.
- Class: dsto.vipa.orbat.ORBATDataType
    - Constraints specified as class wide:
        - role must not be null when ${structureType == "OU_BRICK" || structureType == "OO_BRICK"}
    - Constraints specified per field:
        - structureType
            - Must not be null (empty).
            - The referenced object must be valid.
        - noLinks
            - Must not be null (empty). **Only when the following validation group(s) are being checked:** dsto.validation.Persistence,
            - Minimum value is 0.
        - noUnits
            - Must not be null (empty). **Only when the following validation group(s) are being checked:** dsto.validation.Persistence,
            - Minimum value is 0.
- Class: dsto.vipa.orbat.ORBATType
    - Constraints specified as class wide:
        - Linked Unit Containment check will be performed.
        - Multiple Command check will be performed.
        - Cyclic Link check will be performed.
        - Multiple Command Root check will be performed.
        - Units Have Association Links check will be performed.
        - Field Value Combination check will be performed.
        - Field Value Combination check will be performed.
        - linkscardMin must be equal to 0
        - linkscardMax must be equal to 1
        - Field Value Combination check will be performed.
        - linkstype must be equal to ASSOCIATION when structure type is OPO
        - noUnits must be equal to the number of units **Only when the following validation group(s) are being checked:** dsto.validation.Persistence,

        - noLinks must be equal to the number of links **Only when the following validation group(s) are being checked:** dsto.validation.Persistence,
        - units.amObjRefsmaximum must not be null when ${structureType == "UE" || structureType == "OPO"}
        - units.amObjRefs.maximumcapabilityType must be equal to OLOC in a UE
        - units.amObjRefsminimum must not be null when ${structureType == "UE"}
        - units.amObjRefs.minimumcapabilityType must be equal to MLOC in a UE
        - units.amObjRefsrequired must not be null when ${structureType == "UE"}
        - units.amObjRefs.requiredcapabilityType must be equal to FTE in a UE
        - units.amObjRefsamRef must have at least one element
        - units.amObjRefsamRef[0].capabilityType must be equal to LE in a UE
        - units.amObjRefs OLOC must be greater than or equal to MLOC
        - linkscardMax must be greater than or equal to 0
        - units force element must be of type BRICK
        - units force element must be of type BRICK
        - units force element must be of type INSTANCE
        - units must be empty for type OO
        - units must be empty for type OO_BRICK
        - units contains a deprecated entity **Only when the following validation group(s) are being checked:** dsto.validation.Deprecation,
    - Constraints specified per field:
        - links
            - There must be no duplicate entries.
            - The referenced object must be valid.
- Class: dsto.vipa.geo.PositionType
    - Constraints specified per field:
        - name
            - Size must have minimum value of 0 and maximum value of 255.
        - geoid
            - Size must have minimum value of 0 and maximum value of 255.
        - locationName
            - Size must have minimum value of 0 and maximum value of 255.
        - countryName
            - Size must have minimum value of 0 and maximum value of 255.
        - lat
            - Value range has minimum value of -90 and maximum value of 90.
        - lon
            - Value range has minimum value of -180 and maximum value of 180.
- Class: dsto.vipa.commons.QTimeType
    - Constraints specified per field:
        - start
            - Must not be null (empty).
- Class: dsto.vipa.orbat.RelType
    - Constraints specified as class wide:

        - cardMax must be greater than cardMin
    - Constraints specified per field:
        - type
            - Must not be null (empty).
        - cardMin
            - Minimum value is 0.
        - cardMax
            - Minimum value is 0.
- Class: dsto.vipa.orbat.UnitDataType
    - Constraints specified as class wide:
        - role must not be null when ${type == "BRICK"}
        - noAmObjRef Has no Aide Memoire references assigned. **Only when the following validation group(s) are being checked:** dsto.validation.AMReferences,
        - amObjRefs OLOC must be greater than or equal to MLOC
    - Constraints specified per field:
        - type
            - Must not be null (empty).
            - The referenced object must be valid.
        - position
            - The referenced object must be valid.
        - noAmObjRef
            - Must not be null (empty). **Only when the following validation group(s) are being checked:** dsto.validation.Persistence,
            - Minimum value is 0.
- Class: dsto.vipa.orbat.UnitType
    - Constraints specified as class wide:
        - noAmObjRef noAmObjRef must be equal to the number of amObjRefs **Only when the following validation group(s) are being checked:** dsto.validation.Persistence,
    - Constraints specified per field:
        - amObjRefs
            - The referenced object must be valid.
- Class: dsto.vipa.orbat.UnitTypeAmObjRef
    - Constraints specified per field:
        - amid
            - Must not be null (empty).
            - Size must have minimum value of 1 and maximum value of 255.
        - name
            - Must not be null (empty).
            - Size must have minimum value of 1 and maximum value of 100.
        - type
            - Must not be null (empty).
        - version

    - Minimum value is 0. **Only when the following validation group(s) are being checked:** dsto.validation.Deprecation, dsto.validation.AMReferences,
- maximum
    - The referenced object must be valid.
- minimum
    - The referenced object must be valid.
- required
    - The referenced object must be valid.
- amRef
    - The referenced object must be valid.

DEFENCE SCIENCE AND TECHNOLOGY GROUP
DOCUMENT CONTROL DATA

1. DLM/CAVEAT (OF DOCUMENT)

2. TITLE

The Vital Planning and Analysis (ViPA) ORBAT Data Service Architecture and Design Overview

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (U/L) NEXT TO DOCUMENT CLASSIFICATION)

Document (U)
Title (U)
Abstract (U)

4. AUTHOR(S)

Kyran Lange

5. CORPORATE AUTHOR

Defence Science and Technology Group
West Avenue
Edinburgh SA 5111

6a. DST Group NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DST-Group-TN-1539	AR-016-657	Technical Note	July 2016

8. Objective ID	9. TASK NUMBER	10. TASK SPONSOR
AV12780643	07/204	SLB

13. DOWNGRADING/DELIMITING INSTRUCTIONS

14. RELEASE AUTHORITY
Chief, Joint Operations and Analysis Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT
Approved for public release.

16. DELIBERATE ANNOUNCEMENT
Approved for public release.

17. CITATION IN OTHER DOCUMENTS Yes

18. RESEARCH LIBRARY THESAURUS
Order of Battle, Strategic intelligence, Web services, Technology

19. ABSTRACT

The Vital Planning and Analysis (ViPA) workbench is an automated logistics feasibility analysis tool used to support planning and logistics. ViPA makes use of Order of Battle (ORBAT) data as an input for its calculations. The ORBAT Data Service is a Web Service for storing force structures to be used by the ADF. The service is composed into what is known as a Service Oriented Architecture which provides a loose coupling of self-contained services. This architecture helps to provide a reusable, access controlled and resilient data source. This report describes the design of the ORBAT Data Service and how it is used to manage and share ORBAT data between tools such as ViPA.